IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:	)	
	)	
Michael Thess and Andreas Ittner	)	
	)	Art Unit: **Unassigned**
Serial No. **10/673,040**	)	
	)	Examiner: **Unassigned**
Filed: **September 26, 2003**	)	
	)	
For: **Method and Apparatus for Determining a Set of Large Sequences From an Electronic Database**	)	

CLAIM OF FOREIGN PRIORITY AND TRANSMITTAL OF ENGLISH TRANSLATION OF APPLICATION AND CERTIFIED COPY OF FOREIGN PRIORITY DOCUMENT

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Pursuant to 37 C.F.R. §1.55, Applicant submits herewith a full English translation of the above referenced patent application, which is also an English translation of the priority document, and a certified copy of German Patent Application 10245859.6 filed in the German Patent Office on September 30, 2002 and the basis for Applicant's foreign priority in this application under 35 U.S.C. §119. With submission of this certified copy, Applicant respectfully claims the benefit of the filing date of German Patent Application 10245859.6 under 35 U.S.C. §119 for the above-referenced U.S. Patent Application.

Applicant and the undersigned verify that the accompanying English translation of the above referenced application is a true and accurate English translation of the application and the German priority document.

Certificate of Mailing

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on January 26, 2003.

Peter G. Pappas - Reg. No. 33205

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine, or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of any patent issuing on this application.

Respectfully submitted,

By: Peter G. Pappas
Reg. No. 33205

SUTHERLAND ASBILL & BRENNAN LLP
999 Peachtree Street, NE
Atlanta, Georgia 30309-3996
Telephone: (404) 853-8000
Facsimile: (404) 853-8806

Attorney Docket No.: 17346-0009

Method and apparatus for determining a set of large sequenc s from an electronic data base

The invention relates to the art of sequential analysis in electronic data bases.

Background of the invention

In sequential analyses sequences of items are extracted from transactional electronic data. As more and more transactional data are acquired electronically sequential analysis becomes more important.

A practical example of a sequential analysis is the clickstream analysis for calculating typical user paths of Web site users. Further examples of sequential analyses are text analysis for extracting characteristic series of words in electronically memorized documents and the analysis of baskets of merchandise taking into account the order of products in trade to define typical product chain purchases. Moreover, sequential analyses are widely used in chemistry and genetics.

A great number of specialized methods of sequential analysis are known to date, especially in genetics. However, very few of them are universally applicable. In the simplest case, all variants of possible sequences are studied as regards their frequency. But for reasons of computer capacity that can be done only for small amounts of data. Sequential analysis algorithms based on search trees present an alternative, such as the algorithm "Capri" by Messrs. Lumio (http://www.spss.com/PDFs/CP2SPCZ-1201.pdf). Yet their speed, too, is insufficient for analyzing large transaction data. Methods based on the classical analysis of baskets of merchandise offer a quicker alternative and have already been used for sequential analysis of baskets of merchandise (Agrawal R., Srikant R., Mining Sequential Patterns. IBM Almaden Research Center, 650 Harry Road, San Jose).

The invention

It is an object of the invention to indicate a method of and an apparatus for determining a set of large sequences from an electronic data base by which known shortcomings are overcome and, especially, an efficient sequential analysis can be performed on large amounts of data in diverse applications.

According to one aspect of the invention, a method is provided of determining a set of large sequences from an electronic data base comprising a set D = {d1,...,dn} of transactions di ($1 \leq i \leq n$) in a computer system with an implemented query module, each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej ($1 \leq j \leq m$) and the method comprising the following steps:

a) determining a set L1 of large sequences from the set D of transactions, the large sequences of set L1 each comprising exactly one item of the record E, and an associated support value S_{L1} on the sequence D of transactions each being greater than or equal to the given support value S;

b) determining a set L2 of large sequences from the set D of transactions, the large sequences of set L2 each comprising exactly two items of the record E in a respective order R_{L2}, and an associated support value S_{L2} on the set D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising one of the large sequences of set L1, as a partial sequence, being taken into account in determining set L2;

c) determining a set Lk (k>2) of large sequences from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{LK}, and an associated support value S_{LK} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the

respective order R_{Lk-1}, being taken into account in determining set Lk; and

d) repeating step c) for k = k+1 and terminating the repetition of step c) when a given termination condition is fulfilled.

According to another aspect of the invention an integrated sequential analysis system is provided, comprising:

- an electronic data base comprising a set D = {d1,...,dn} of transactions di ($1 \leq i \leq n$), each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej ($1 \leq j \leq m$);
- a query module comprising a query means coupled to the data base and a processing means for detecting query parameters and generating queries to the query means;
- means for determining a set L1 of large sequences from the set D of transactions, the large sequences of set L1 each comprising exactly one item of the record E, and an associated support value S_{L1} on the sequence D of transactions each being greater than or equal to the given support value S;
- means for determining a set L2 of large sequences from the set D of transactions, the large sequences of set L2 each comprising exactly two items of the record E in a respective order R_{L2}, and an associated support value S_{L2} on the set D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising one of the large sequences of set L1, as a partial sequence, being taken into account in determining set L2;
- means for determining a set Lk (k>2) of large sequences from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{Lk}, and an associated support value S_{Lk} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the respective order R_{Lk-1} being taken into account in determining set Lk; and

- means for repeating step c) for k = k+1 and terminating the repetition of step c) when a given termination condition is fulfilled.

In contrast to the classical basket of merchandise analysis, the invention takes into account the order of items in the transactions. The invention permits automatic evaluations of extensive amounts of electronic data according to characteristic sequences to be made efficiently in terms of time and computer capacity. Sequential analysis of large amounts of data, such as contained in logfiles of heavily frequented Web servers thus becomes possible for the first time.

Drawing

The invention will be described further, by way of example, with reference to the accompanying drawings, in which:

Fig. 1 is a diagrammatic presentation to explain a sequential analysis method according to the invention in connection with a clickstream analysis;

Fig. 2 is a diagrammatic presentation of an arrangement for carrying out a sequential analysis method according to the invention;

Fig. 3 is a flowchart of an embodiment of a sequential analysis method according to the invention in accordance with a first variant; and

Fig. 4 is a flowchart of an embodiment of a sequential analysis method according to the invention in accordance with a second variant.

Examples of embodiments

A method of and an apparatus for determining a set of large sequences from an electronic data base will be described below with reference to figs. 1 to 4.

First of all, the problem to be solved by sequential analysis will be explained in detail. Let it be assumed that E = {e1, e2,..., em},

a record of literal constants, referred to as items. A sequence $<s_0, s_1, \ldots, s_k)$ be an ordered list of items. D = {d1, d2,..., dn}, a record of transactions, each transaction di ($1 \leq i \leq n$) a sequence of items from E so that: di $\subseteq$ E. An unambiguous identifier, designated "di-ID", is associated with each transaction.

The sequence <a0, a1,..., ak> is included (partial sequence) in another sequence <b0, b1,... b1> when natural non-negative numbers $i_1 < i_2 < \ldots < i_k$ exist so that:

$a1 = bi_1$; $a2 = bi_2$;...; $ak = bi_k$.

It is defined that a sequence A possesses a support s% on the set of transactions D if it is contained in s% of all transactions of set D. A sequence is said to be large if its support is not smaller than a given minimum value. A sequence having a length k is called k-sequence. The set of all k- sequences will be referred to below as Lk. The set of all k- sequences which potentially may be large is called Ck.

Set E can be mapped unambiguously on a subset of natural numbers, i.e. an unambiguous natural identifier may be assigned to each item from E.

Using the designations introduced above, the object to be met may be defined as follows: Based on the given sets E and D, all sequences are to be found whose support value is greater than or equal to a given minimum value, i.e. the following set is looked for:

$$L = \bigcup_k L_k$$

A known solution trial to resolve the above problem resides in listing all the possible sequences of items of set E and counting the transactions containing them. The number of all possible sequences of set E having a length of from 1 to k is found. The power of set E is m. The number of sequences of length 1 equals the number of items of set E, in other words m. The number of all possible sequences of

length i of m items is m^i. Then the number of all sequences N equals the sum of the geometric progression to the basis m:

$$N = \sum_{i=1}^{k} m^i = \frac{m^{k+1} - m}{m-1}.$$

For just m = 100 and k = 10 already we get $N = \frac{100^{11} - 100}{100 - 1} \approx 10^{20}$ different sequences. It is obvious that such a solution trial is inacceptable. For real data, distinctly less sequences are contained in all transactions of set D.

It is much more logical and quicker to test all the sequences which are included in each transaction and to define the support value for each of them. Assuming the length of transaction d0 to be m, this would mean that the transaction d0 includes the number of all sequences of length i which equals the set of i combinations of the items above m. Then the total number of sequences N contained in a transaction of length m is:

$$N = \sum_{i=1}^{m} \frac{m!}{i! \cdot (m-i)!} = 2^m - 1$$

A transaction of length 10 encompasses 1023 sequences, a transaction of length 20 encompasses 1048575 sequences of lengths from 1 to 10 and, accordingly, 20. In spite of the fact that the numbers mentioned describe the most unfavorable case, the number of possible sequences contained in a transaction may become very large.

Each of those sequences must be tested for its minimum support condition, in other words the sum of transactions including them must be formed. It is evident that this process takes a lot of time, too. For this reason, the methods according to the invention described here make use of a solution trial for which no superfluous tests are needed.

Two variants of extracting large sequences will be described below. Both utilize a common record of basic concepts, but they differ in the summing of the support values of the sequences.

The methods belong to the category of iterative procedures. In each step of the procedures all large sequences of a given length are defined. Operation of the procedure is continued until all large sequences of maximum length have been found, in other words until no further large sequences are discovered in the current operating step. Another criterion for terminating the procedure may be the reaching of a maximum length of the sequences which were predetermined by the user.

With both variants of the method according to the invention the large sequences of length k are used for constructing the set of new potential large sequences of length k+1. A sequence is called a candidate if it was generated from large sequences of a length reduced by 1 and if, possibly, it likewise is a large sequence. In this manner the set of candidates having length k presents the set Ck. Obviously, the following is true: $Ck \supseteq Lk$.

1. Method of generating the candidate set.

In both methods, the same principle of generating candidates is applied. The principle differs slightly from the one of generating candidates in the preceding IBM solution trial. The function for generating the candidates uses all large k-sequences - Lk as the input set. The result it offers is a super set of the sets of all large (k+1) sequences - Ck+1. The method of generating the candidate set may be written as stated below, making use of an SQL-like syntax:

INSERT INTO Ck

SELECT $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

FROM p,q

WHERE $p.item_2 = q.item_1$, $p.item_3 = q.item_2, \ldots$ $p.item_k = q.item_{k-1}$.

The mode of generating the candidates may be illustrated as follows.

Assuming the large sequences of length 4 listed below were found in a step of the procedure:

1. 1 3 2 6
2. 1 4 1 2
3. 2 1 3 2
4. 2 2 2 2
5. 3 1 2 1
6. 3 2 6 2
7. 4 1 2 1.

Checking all possible pairs of large 4-sequences, beginning with the first one and including also all those consisting of the same sequence, the following candidate set is obtained:

1 3 2 6 2 - result of the union of 1 and 6
1 4 1 2 1 - result of the union of 2 and 7
2 1 3 2 6 - result of the union of 3 and 1
2 2 2 2 2 - result of the union of 4 and 4

Sequences 5., 6., and 7. did not provide a single pair in which they would have been the first of the "parents". Sequence 5. did not take part in the generation of candidates at all. Sequence 4. formed a union with itself.

The result of the procedure of this method may bring forth a situation where not a single candidate sequence is generated by this method. This means that it is impossible to find but a single sequence of the next level, the k+1 level. Consequently, the method ends with this step.

In the example described above, all possible pairs of large sequences of the previous level were drawn upon for constructing the candidate set. When the method works on real data, there is great likelihood of finding, in a particular step, a set of all large sequences which may consist of thousands or tenthousands of sequences. The process of completely reviewing all pairs possesses a square complexity. On real data, therefore, millions or hundreds of millions of operations would have to be carried out to compare se-

quences. The process of generating candidates thus might involve some serious time expenditure.

For this reason, a data structure was elaborated which permits to reduce the complexity of reviewing the pairs of large sequences. According to this concept, the first and the last large (k+1) sequence must be stored for each large k-sequence, beginning with the k-sequence under review. Of course, that requires sorting of the record of k-sequences from the small k-sequences to the large sequences. The k-sequence A is defined as being smaller than the k-sequence B if the first different item is smaller in sequence A than in sequence B.

Thus, we possess in a step of the method Lk - the set of the large k-sequences of the preceding step, Ck+1 - the set of the candidates. For each k-sequence from Lk, moreover, the number of the second "parent" which generated this sequence is memorized. Based on this number, the spectrum of sequences from Lk with which each k-sequence may be united can be defined for each k-sequence. This data structure is employed in both variants of the method specified. A dynamic array having the dimension |Lk-1|x2 is used for memorizing the numbers required for the sequences from Lk-1.

In the k^{th} step, the set of large sequences Lk is available with each of which the number of the sequence from Lk-1 is connected that forms the second "parent". As a result of the work the set Lk+1 is obtained from Ck+1 in the k^{th} step, and for the sequences of this set the second parent's numbers are memorized, and a new structure is formed of the numbers of the sequences from Lk.

In the example described above, the sequence <2 1 3 2> was formed in the preceding step from the sequences <2 1 3> and <1 3 2>. Assuming sequence <1 3 2> had had its place at number 3. of the list of sequences L3, then the numbers 1 and 1 must be memorized in the structure described for the sequence of number 3. - in accordance with the first and last sequences of L4 which begin by <1 3 2>. If the new sequence <2 1 3 2 6> proved to be large the number of its second

parent is memorized - 1, for sequence <2 1 3 2> it is number 3 - the number of the first newly obtained 5-sequence.

If a descendant does not exist for a sequence from Lk-1 this fact is memorized in the structure of the numbers by way of an invalid number - for example 1.

2. The first variant

The process of operation of the first method may be subdivided into a plurality of stages:

(i) search for all large 1-sequences - the various items of E - L1.

(ii) search for all large 2-sequences - L2 on the basis of L1. For each large sequence found, the list of transaction numbers containing the same is memorized at the same time.

(iii) search for the large k-sequences on the basis of Lk-1. The third stage is repeated until Lk = ∅ has been obtained.

The first variant for which a flowchart is shown diagrammatically in fig. 3 may be represented as follows in pseudo code:

```
L1 = {large 1-sequences};
C2 = candidate-gen(L1);          // 2 candidates
forall transactions t ∈ D do begin
     Ct = subset(C2,t);    // candidates contained in t
     forall candidates c ∈ Ct do
          c.count++;
end
L2 = {c ∈ C2| c.count ≥ minsupp};
for (k=3; Lk-1 ≠ ∅; k++) do begin
     Ck = candidate-gen(Lk-1);
     forall candidates c ∈ Ck  do begin
          Tc = subset(c,D);     // transactions including c
          c.support = |Tc|;
     end
     Lk = {c ∈ Ck | c.count ≥ minsupp};
end
answer = ULk;
```

In the first variant of the method, two solution trials are combined for counting the support values of the candidate sequences. Sequential transit through the set of input transactions is the method employed to find set L2. For each transaction, the candidates con-

tained in it are defined, and for each candidate the counter of the support value is incremented by one.

In the first variant of the method the same solution trial was selected for this stage and for the next one: for each candidate, the list of transactions is determined which might contain the candidate. The list of transactions results as the cut set across the memorized transaction lists of the parents of the candidate under investigation. Evidently, the candidate sequence can be included only in those transactions in which both parents of the particular candidate are memorized. The list of transactions of the candidate includes the numbers of the transactions and, making use of the cut set formation, only those numbers are added which are included in the lists of transactions of both its parents. The list thus obtained permits the number of transactions which must be tested for the presence of a certain candidate to be reduced substantially, especially so in the final steps of the procedure.

However, transit through the whole set of candidates is a precondition for the statement of the transaction lists. In the second stage of the method, when counting set L2, nothing but the items of set L1 are known. Any pair of large 1-sequences forms a candidate of length 2. During operation of the method with real data, the power of set L1 may reach several tens of thousands of 1-sequences. The number of candidates of length 2 then will be a few hundred million. A complete review of all the possible candidates would require an unreasonable amount of time. In this case it makes sense to select the principle which was used fully in the second variant of the method.

According to the basic idea of the second variant of the method, a counter of the support value is introduced for each candidate. That is followed by the sequential transit through the entry set of the transactions. For each transaction, the candidates it includes are defined, and their counters of the supports are incremented by one. In the second variant of the method, a more complex scheme is used for defining the candidates which are contained in a particular transaction, but a simpler method may be resorted to for defining set L2. Two transits through the transactions, one embedded in the

other, are required for defining the numbers of all candidates. In the outer cycle, the large items contained in the transaction are selected one after the other. The inner cycle begins with the item which succeeds the item chosen in the outer cycle. Each selected pair of large items of set E represents a length-2 candidate and, therefore, the corresponding counter of the support value of the candidate must be incremented. The number of the counter is calculated starting from the indices of the selected items in the list of large items and the total number of large items found. Let a and b - the respective selected first and second items of a candidate and N - be the number of 1-sequences found in the preceding stage of the method - the large items from L1. Then the number of the respective candidate is obtained according to the following formula: $n_c = Na+b$

If the repeated incrementing of the counter of the support of a specific candidate is to be excluded, in dealing with one and the same transaction, a means of marking the fact that incrementation of the counter took place in a particular transaction must be provided. The simplest way of doing that is to store, for each candidate, not only its counter of the support but, in addition, also the number of the last transaction which caused the counter of the candidate of interest to be incremented. In view of the fact that the transactions from the entry set are dealt with successively, the content of the field of the number of the last transaction gives an indication whether the counter was incremented since the beginning of dealing with the concrete transaction under review.

The manner of proceeding as described above for determining the 2-sequences L2 has the advantage that the calculations involved are relatively simple because only those candidates are processed which indeed are contained in the transactions input. Yet this method also suffers from a serious disadvantage. For each candidate, two values related to it must be stored: the counter of the support value and the number of the last transaction which incremented the counter. Using a 32 bit word for each field which, theoretically, would allow processing a maximum of 2147483647 transactions, requires 8 bytes per candidate. At the stage of defining set L2 there may be hundreds of millions of different candidates. This stage in the operation of

the method consequently requires some hundred million bytes of RAMs and that, indeed, presents a serious limitation. It is the square-law dependence of the memory needed on the number of large 1-sequences found in the preceding stage that causes this serious limitation.

To overcome this limitation, a scheme of multiple transit of the processing of the transaction set in the course of constructing set L2 was realized in the final version of the first method. During a full transit through the input set of transactions, only those pairs are dealt with whose first item lies within a certain range of values. In this manner the storage space required for the second stage of the procedure can be restricted, in consideration of the storing capacity available in the computer. Of course, greater memory restrictions in turn will demand more transits and, accordingly, the procedure will become slower in the second stage. The method described of defining the 2-sequences L2, therefore, is a compromise between the necessary storage space and the processing speed. This fact is confirmed by the experimental execution time behavior of the method.

2.1 Further implementation details of the first variant

In the first stage of operation of the method all items of set E must be found that occur in the input transactions no less than a predetermined number of times. As the numbers and frequencies of the various items from E are not known initially a method must be provided of storing both the actual item and the number of transactions in which it is contained. When practicing both variants of the method, a transit through all the input transactions takes place. A standard class (e.g. from Java) - the *hash table*, is drawn upon for storing the items and their counters. This class permits storing of the pair <key, value>, and for quick search of the values it uses the mechanism of the hash table in respect of the key values. In implementing the method, the key chosen is an object of the *integer* class (wrapper class for integral values) (e.g. from Java) which is initiated with the value of the selected item, and the value chosen is another object of the *integer* class which is initiated with the value of its counter. When selecting another item of a transaction,

access is taken to the object of the *hash table* class to obtain the value through the key which is identical with the selected item. If such a value was found in the hash table, another rewrite, incremented by one, is caused. If no such value was found for the key, the pair to be registered in the hash table will be the pair of which the value it contains is the initial value of the counter, in consideration of the item already selected - 1.

Repeated incrementing of the counter of a transaction is avoided, in implementing the method, by choosing a different class (e.g. from Java) - *vector*, in which all those items are stored that were already processed by a transaction. Before searching for the value in the hash table, the method attempts to find the item in the instance of the *vector* class. If such an item was found, it means that it was processed before in this transaction, and the method will proceed to the next item of the transaction. Of course, the vector of values is purged when the method proceeds to the next transaction. Having gone through the transaction set, only those items are selected whose counter values are not smaller than the minimum value. The large items found are stored in the array and sorted so that, in the further course of proceedings, binary search can be applied to the items and be mapped to the set of integral positive numbers 0,..., N-1, with N being the number of large items found. The large items are used to form set L1 of the large 1-sequences.

In implementing the method, the *TransactionList* class is used for storing the list of the numbers of the transactions. It permits storing the numbers of the transactions in an array, adding a number to the end of the list, provided the numbers are added in rising order. This is an indispensable condition for the correct operation of the method of the *TransactionList* class which forms the cut set of both transaction lists. The initial lists of the transactions are formed in the course of the sequential transit through the set of input transaction during the stage of the construction of L2, at the same time, establishing, in a natural way, the conditions of the sorting order. Further, the new lists are created while the cut set operation is executed. The lists of the numbers are stored in an array. However, since the length of the array is defined during the

stage of dynamic storage allocation, additional fields of the array must be made available, as required, by increasing the length. An array having an initial size is made up and gradually filled during the generation of an instance of the *TransactionList* class. When the maximum of numbers is reached, the memory is made available once more. The maximum storage capacity of the array is increased tenfold, for instance, a new array having the same maximum length is made available and the contents of the previous array are copied into the same. The old array may be marked as superfluous (e.g. for the virtual machine in Java).

During the third stage of operation of the method, all the large k-sequences memorized during the preceding step are reviewed. The numbers of the first and last large k-sequences, with which the sequence found can be united, are found by way of the stored number of the second parent of the sequence, making use of the data structure described above. Moreover, the intersection forming operation is carried out across the lists of transactions of both sequences. If the length of the resulting list is smaller than the required minimum support value the candidate under review is discarded immediately even without an operation of a direct candidate and counting of the number of transactions containing the same. Otherwise, the sequence of the items of the candidate itself is generated and each transaction of the list obtained from the formation of the intersection is checked as to the content of candidate sequences it includes. If a corresponding sequence was found in a transaction the number of the transaction is memorized. In this manner, at the end of processing a candidate, there will be a list of transactions which include the candidate. This candidate will be stored together with the number of its second parent if the number of such transactions is not smaller than the minimum number.

The actions described above are repeated for each candidate. If, ultimately, not a single large sequence was found the operation of the third stage of the method is terminated and the results are output via the interface. There may be two reasons why not a single sequence was found in the course of one step. In set Lk, there may not be a sequence which might form a candidate together with another se-

quence, i.e. not a single candidate exists in a step. Another reason may be the variant where, ultimately, not a single candidate presents a large sequence.

Summarizing, the first variant of the method proves to be quick on large amounts of data, and its storage space requirement is small. As far as storage space and computing time regarding the use of transaction lists are concerned, empirical calculations and experimental results show that, based on real data, they tend to be greatly reduced because, as the sequences being processed become longer they will be contained in a progressively smaller number of transactions, as a result of which not only the lists become shorter but also the amount of time needed to process them. Consequently, it is a requirement for operation of the first variant of the method that the initial set of transactions be entered in the form of a list of sequences of integral numbers, the minimum support value be input as a floating value, and the storage space available for the second stage of the method be input, too.

3. *Second variant*

The second variant of the method, of which a flowchart is illustrated diagrammatically in fig. 4, was elaborated and implemented for purposes of comparison with the first variant of the method as regards speed and storage space. The second variant can be represented as follows in pseudo code:

```
L1 = {large 1-sequences};
for (k=2; Lk-2 ≠ ∅; k++) do begin
      Ck = candidate-gen(Lk-1);  // new candidates
      forall transactions t ∈ D do begin
            Ct = subset(Ck, t);
            forall candidates c ∈ Ct do
                  c.count++;
      end
      Lk = {c ∈ Ck | c.count ≥ minsup}
end
answer = U Lk;
```

The method illustrated is very general and does not implement the process of selecting the candidates that are included in a particular transaction. In the simplest case this may be reviewing all can-

didates with the aim of selecting those which are contained in the sequence of the transaction. Another variant provides for reviewing all sequences of a given length and selecting those from among them which represent candidates. It is evident that both methods are disadvantageous for being inefficient. Therefore, a different method is described for implementing the second variant.

The second variant makes use of quite a few of the same ideas as the first one. In particular, the same principle is applied without much alteration for calculating set L1. The entire description given above of the first stage of operation of the first variant of the method applies to the second variant as well. As already mentioned, for adding the support of the candidates, the second variant of the method makes use of the concept which, in the first variant, was applied only in the second stage for generating the set of large 2-sequences, L2.

3.1 Further implementation details of the second variant

A specific data structure is used with the second variant of the method so as to have the possibility of defining the list of candidates included in it, for every transaction. From the second stage of the method on, each transaction is presented as a list of the numbers of the k-sequences found in the preceding step of the method. Reviewing the pairs of these k-sequences and uniting them according to the method described above of generating candidates, all the candidates of length k+1 can be obtained that are included in a particular transaction. Yet the list of the large k-sequences is organized as a two-dimensional array of values so as to separate the candidates. The resulting data structure will be illustrated in the example below.

Let us assume the entry set of transactions includes the following transaction:

<3 1 3 2 1>.

Furthermore, let it be assumed that in the second stage of operation of the method it was discovered that the sequences <3 1>, <3 2>, <1 2>, and <1 1> contained in that transaction are large, having num-

bers 5, 7, 3, and 1, respectively. In the second stage of operation of the method the following data structure was generated, which corresponds to the given transaction:

1. 5 7
2. 3 1
3. 7 5
4. -

The number of lines of this structure equals the number of items in the transaction during the stage of construction of set L2. In the further stages this set equals the number of lines in the preceding structure for this transaction. Each line contains the numbers of the large sequences which begin with this item or with this line in the structure that was generated in the preceding step. In each step, the pairs of numbers of the large sequences of the previous level are reviewed, and the method searches all pairs of numbers which might be united. The test for possible union is carried out on the basis of the same principle as with the first variant of the method.

A structure of this kind is formed for each transaction and is updated in each step of the method. The first structure is generated in the second stage of the method, and that is the only difference compared to the corresponding stage of the first variant of the method. In one step of the method, all transactions of the entry set are processed successively, utilizing the structure just described. Step by step each item of each line is selected, and following that the items of each subsequent line are searched. If the pair thus selected may become a candidate the counter of its support is incremented by one. To avoid multiple incrementing of the counter during the processing of a transaction, the idea of the first variant of the method may be resorted to in that the number of the last transaction incremented is stored for each candidate.

This illustrates that actually it is not necessary, with the second variant of the method, to generate candidate sequences during the execution time because, contrary to the first method, that is not needed for adding up the support values. The large k-sequences found

during the k^{th} step are stored in the form of the pairs of their parents. That helps save storage space because in practice the large k-sequences may come about at the end of the operation proper of the method during the transit from set L1 to set L2 so that they will not unnecessarily occupy storage space while the main cycle of the method is underway.

The process of filling the required structure may turn out to be difficult because, while the numbers of the candidates are known during the processing of the transaction, those of the large sequences are not as they are defined only after all transactions have been processed. During the processing, therefore, the numbers of the candidates contained in them are entered into the renewed structure. When all transactions of the entry set have been processed the numbers included in the structure must be corrected to allow for the candidates which did not fulfill the support condition. The part of the method during which the numbers are corrected may be presented as follows:

```
m = 0;      // accumulator of the correction
forall c∈ Ck do begin      // for each candidate
     if c.count>=minsup then c.correction = -m-1
     else m++;
end
```

Having accomplished this fragment of the method, the value of the correction is known for each candidate and must be deducted from the number of the candidate so as to provide the number of the large sequence. In fact, to get the number of the sequence from the number of the candidate the number of candidates that did not make the test must be deducted whose numbers are smaller than the number of the candidate in question. The final stage of the renewal of the structure of the transactions is the transformation of the numbers of the candidates into the numbers of the large sequences simultaneously with the removal of the numbers of the non-large candidates and the sorting.

The sorting serves to expedite the search for paired sequences during the processing of the transaction structures, in other words those sequences with which the selected sequence may be united.

Without sorting of the lines of the structure of the transactions the search for the second parent would involve a full transit of the subsequent lines, including a test of the condition for union. Sorting the values of the numbers of the large sequences introduces a rising order, thus providing the opportunity of most rapidly finding a value in the line, by binary search, starting with which the numbers of the parent sequences are memorized. Moreover, an orderly transit through the line is carried on until a number of a sequence is larger than the maximum parent number for the selected first parent. Experimental data have demonstrated that the sorting introduced, which is not absolutely required for the operation of the method, accelerates operations in those cases in which the structure related to a particular transaction contains some very long successive lines. With the first variant, without sorting, a clear drop in efficiency could be observed in some parts of the transaction set, which is why the overall time of the operation of the method is much prolonged. Having applied the sorting, the method operated many times faster on such data, depending on the particular entry transaction sets.

During the processing of the transaction structures, the numbers of the large sequences found in the preceding step of the method and included in the transaction currently being processed are selected successively. For each number selected it becomes a task to define the numbers of the first and last parent with which the sequence having the selected number may be united. Besides, the number must be known for every candidate in order to be able to increment the counter. This problem is solved partly as was done with the first variant of the method. Yet the implementation of the second variant makes use of a slightly modified idea in consideration of the requirement that the number of the candidate must be known. In the case of the first variant, the number of the candidate is known because, for all practical purposes, in the main cycle a transit through the candidate set takes place and a specific structure is renewed in each step for quickly determining the first and last numbers of possible parents. With the second variant, a modified structure is applied in consideration of the need for storing the numbers of the corresponding candidates. In each entry of this structure,

not only the numbers of possible parents are stored but also the number of the first candidate that can be generated. Via the number of the second parent of a large sequence, thus the numbers of the parents can be determined with whom the sequence under review may be united, and also the numbers of the corresponding candidates.

With the second variant of the method as, by the way, also with the second stage of the first method, storage space must be allocated to the values of the counter of the support and the array of the numbers of the transaction which was the last to increment the counter.

4. Description of a program interface

The program interface is considered in the form of an example of applying the method in a Java program.

The *TransactionSetSeq* class is used for storing the record of the sequential entry transformations. An instance of the class is realized by the following instruction:

```
TransactionSetSeq tss = new TransactionSetSeq();
```

Now we add all transactions of the entry record:

```
for (int i=0;i< <number of transactions>;i++) {
```

Each sequential transaction is represented as an instance of the *ItemSetSeq* class:

```
ItemSetSeq iss = new ItemSetSeq();
```

The items of the transactions are integral numbers - the following method of the *ItemSetSeq* class serves to add a new item to the transaction:

```
void addItem(int item);
```

We add all the items of the current transaction:

```
for (int j=0;j< < number of items of the current transaction>;
j++)
    iss.addItem(<number of the item >);
```

The following method of the *TransactionSetSeq* class is used for adding a transaction to the transaction record:

void addTransaction(ItemSetSeq iss);

tss.addTransaction(iss);

}

An instance of the *Sequential* or *Sequential2* class must be generated for calculating the sequences:

Sequential seq = new Sequential();

Sequential2 seq2 = new Sequential2();

The following method contained in the *Sequential* and *Sequential2* classes must be called to start the method:

SequentialResult SequentialAlg(TransactionSetSeq tss, double minsupp);

where

tss - entry set of the transactions,

minsupp - minimum support value.

SequentialResult seqResult = seq.SequentialAlg(tss,minsupp);

or

SequentialResult seqResult = seq2.SequentialAlg(tss,minsupp);

The result of operation of the method is stored in the object of the *SequentialResult* class. The *SequentialResult* class contains the following method for accessing the sequences found:

ItemSetSeqList getLargeSequences();

ItemSetSeqList issList = seqResult.getLargeSequences();

The *ItemSetSeqList* class contains the methods:

int getSize(); - supplying the number of sequences,

ItemSetSeq getItemSetAt(int index); - supplying the sequence of the number index.

for (int i=0;i < issList.getSize(); i++) {

```
ItemSetSeq iss = issList.getItemSetAt(i);
```

The *ItemSetSeq* class contains the following method for accessing the sequence:

```
int getItemAt(int index);
```

The following method supplies the length of the sequences:

```
int getSize();
for (int j=0;j<iss.getSize();j++)
    int item = iss.getItemAt(j);
}
```

The following method of the *ItemSetSeq* class supplies the value of the counter for the support:

```
int getSupportCount();
```

The actual support value *support* for each sequence is the quotient of the value of the support counter and of the number of all transactions.

5. Example: Clickstream analysis

Referring to figs. 1 and 2, the application of the method described above will now be explained by way of example with reference to a clickstream analysis of Web servers.

The user needs a Web browser 10 to call a Web site via the internet, for example the Netscape Navigator, by which he connects to a Web server 20, for example the Apache Web server. Information about HTML pages of a Web site that were fetched is stored by Web servers in data bases 30 or log files 40. The information concerning those HTML pages is stored in a table. The columns correspond to the information about the page that was fetched, e.g. date and time, URL of the HTML page fetched, referer URL, type of Web browser. The lines correspond to the fetched HTML pages themselves. Various mechanisms, not described in any greater detail here, make it possible to identify Web sessions and allocate them to each fetched page. A Web ses-

sion is a sequence of HTML pages which a user has fetched successively from the Web site.

It is the aim of the clickstream analysis to find out typical paths along which users of a Web site mainly move. For instance, it may happen with an E-commerce shop that many users enter the shop through the main page, go into a certain product category, look at a certain product in a detail view, and thereafter leave the shop. The conclusion may be drawn that the product looked at did not satisfy the demand, and that consequently its presentation absolutely needs to be improved. Where users choose cyclical paths, often the conclusion may be drawn that for them the shop is not structured clearly, and so on. Now, having provided a data access module 50, a data mining application 60 implemented via a program and performing the actual clickstream analysis takes access to the data of the HTML pages fetched and to be found either in the data base 30 or in the log files 40. The data access module 50 executes various transformations, like the resolution of IP addresses, recognition of date formats, and the extraction described above of Web sessions.

The mining application 60 calculates the most frequent paths followed by users through the Web site. Finding the typical paths in Web sites is a task for sequential analysis. In this case E = {e1, e2, ..., em} is the set of all HTML pages of a Web site. Each transaction di corresponds to a Web session, the items of which form a sequence <s0, s1, ..., sk> of HTML pages. The total number of transactions D corresponds to the total of Web sessions of the Web server. The clickstream transactions for a simple example are shown at 80(after mappings 90 and 100 into the range of integers) and 110, respectively. It is the task of the clickstream analysis to find all the large sequences of HTML pages which provide the mining results 70, with a given minimum support parameter s. In the example of the clickstream table 80, the large sequences found are shown at 120 or 130, respectively, (after re-mapping) for a minimum support value of 50 %. These are the typical users paths of the Web site whose analysis, as explained above, may be used for improving the structure of the Web site. In particular, the results may be incorporated di-

rectly in the Web server, for instance, by way of recommended HTML pages (recommender systems).

The method described of the invention is very well suited for calculating large sequences, especially so because it can analyze large amounts of data. Log file tables of popular Web sites, in particular, often comprise millions of HTML pages fetched and, therefore, are extremely demanding as regards sequential analysis methods. Consequently, evaluating this information by means of sequential analysis usually was very intensive in terms of computer time and thus limited to special cases (sequences of length 2, only a few Web sessions, etc.). These restrictions are overcome by the method described here for almost any clickstream data occurring on present day Web servers.

The features of the invention disclosed in the specification above, in the claims and drawing may be essential to im-plementing the invention in its various embodiments, both individually and in any combination.

WHAT IS CLAIMED IS:

1. A method of determining a set of large sequences from an electronic data base comprising a set D = {d1,...,dn} of transactions di (1≤i≤n) in a computer system with an implemented query module, each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej (1≤j≤m) and the method comprising the following steps:

a) determining a set L1 of large sequences from the set D of transactions, the large sequences of set L1 each comprising exactly one item of the record E, and an assigned support value S_{L1} on the sequence D of transactions each being greater than or equal to the given support value S;

b) determining a set L2 of large sequences from the set D of transactions, the large sequences of set L2 each comprising exactly two items of the record E in a respective order R_{L2}, and an assigned support value S_{L2} on the set D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising one of the large sequences of set L1, as a partial sequence, being taken into account in determining set L2;

c) determining a set Lk (k>2) of large sequences from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{LK}, and an assigned support value S_{LK} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the respective order R_{LK-1}, being taken into account in determining set Lk; and

d) repeating step c) for k = k+1 and terminating the repetition of step c) when a given termination condition is fulfilled.

2. The method as claimed in claim 1, wherein the set D of transactions is searched for candidate sequences which comprise two of

the large sequences of set L1 or set Lk-1, respectively, as partly overlapping partial sequences, in determining set L2 in step b) and set Lk in step c), respectively, and wherein an assigned support value counter is registered for each candidate sequence found for the first time, and the assigned support value counter is incremented when the respective candidate sequence is determined again in searching the set D of transactions.

3. The method as claimed in claim 1, wherein:

- the set D of transactions is searched for candidate sequences comprising one of the large sequences of set L1, as a partial sequence, when determining set L2 in step b), and an assigned support value counter is registered for each candidate sequence found for the first time, and the assigned support value counter is incremented when the respective candidate sequence is determined again in searching the set D of transactions; and
- a candidate set of sequences is formed, when determining set Lk (k>2) in step c), which set comprises all combinations in pairs of the sequences of set Lk-1, and of the set D of transactions those transactions are defined which comprise at least one sequence of the candidate set, and the support value is determined for those sequences of the candidate set for which a transaction was found that comprised this sequence.

4. The method as claimed in claim 2 or 3, wherein a display value is generated and updated for each assigned support value counter to indicate the transaction for which the assigned support value counter was last incremented.

5. The method as claimed in claim 3 or 4, wherein only those sequences of the candidate set of sequences corresponding to all combinations in pairs of the sequences of set L1 are taken into account of which the first item lies within a given range of values.

6. A computer program product for determining a set of large sequences from an electronic data base comprising a set D = {d1,...,dn} of transactions di ($1 \leq i \leq n$) in a computer system with an implemented query module, each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej ($1 \leq j \leq m$) and the product comprising the following means:

a) means recorded on an electronic storage medium for determining a set L1 of large sequences from the set D of transactions, the large sequences of set L1 each comprising exactly one item of the record E, and an assigned support value S_{L1} on the sequence D of transactions each being greater than or equal to the given support value S;

b) means recorded on the electronic storage medium for determining a set L2 of large sequences from the set D of transactions, the large sequences of set L2 each comprising exactly two items of the record E in a respective order R_{L2}, and an assigned support value S_{L2} on the set D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising one of the large sequences of set L1, as a partial sequence, being taken into account in determining set L2;

c) means recorded on the storage medium for determining a set Lk (k>2) of large sequences from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{LK}, and an assigned support value S_{LK} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the respective order R_{LK-1}, being taken into account in determining set Lk; and

d) means recorded on the electronic storage medium for repeating step c) for k = k+1 and terminating the repetition of step c) when a given termination condition is fulfilled.

7. An integrated sequential analysis system, comprising:

- an electronic data base comprising a set D = {d1,...,dn} of transactions di ($1 \leq i \leq n$), each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej ($1 \leq j \leq m$);
- a query module comprising a query means coupled to the data base and a processing means for detecting query parameters and generating queries to the query means;
- means for determining a set L1 of large sequences from the set D of transactions, the large sequences of set L1 each comprising exactly one item of the record E, and an assigned support value S_{L1} on the sequence D of transactions each being greater than or equal to the given support value S;
- means for determining a set L2 of large sequences from the set D of transactions, the large sequences of set L2 each comprising exactly two items of the record E in a respective order R_{L2}, and an assigned support value S_{L2} on the set D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising one of the large sequences of set L1, as a partial sequence, being taken into account in determining set L2;
- means for determining a set Lk (k>2) of large sequences from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{LK}, and an assigned support value S_{LK} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the respective order R_{LK-1}, being taken into account in determining set Lk; and
- means for repeating step c) for k = k+1 and terminating the repetition of step c) when a given termination condition is fulfilled.

Abstract of the disclosure

Method and apparatus for determining a set of large sequences from an electronic data base

The instant invention relates to a method of and an apparatus for determining a set of large sequences from an electronic data base comprising a set D = {d1,...,dn} of transactions di ($1 \leq i \leq n$) in a computer system with an implemented query module, each of the large sequences on the set D of transactions di having a support value greater than or equal to a given support value S, each of the transactions di of the set D being a sequence of items of a record E = {e1,...,em} of items ej ($1 \leq j \leq m$). A set Lk ($k>2$) of large sequences is determined from the set D of transactions, the large sequences of set Lk each comprising exactly k items of record E in a respective order R_{LK}, and an associated support value S_{LK} on the sequence D of transactions each being greater than or equal to the given support value S, and nothing but sequences comprising two of the large sequences of set Lk-1, as partly overlapping partial sequences, with the respective order R_{LK-1}, being taken into account in determining set Lk.

Clickstream table

80

Websession	HTML page	Position
1059877799704	index.html	1
1059877799704	product.html	2
1059877799704	catalog.html	3
1087643555703	about.html	1
1087643555703	product.html	2
1087643555703	order.html	3
1168740073550	index.html	1
1168740073550	about.html	2
1168740073550	product.html	3
1168740073550	order.html	4
1168740073550	about.html	5
26795398458245	about.html	1
26795398458245	order.html	2
26795398458245	about.html	3

Mapping Web session

90

Websession	di-ID
1059877799704	1
1087643555703	2
1168740073550	3
26795398458245	4

Mapping HTML pages

HTML-Seite	I
index.html	1
about.html	2
product.html	3
catalog.html	4
order.html	5

100

Clickstream table after mapping of Web sessions and HTML pages

TID	T
1	1 3 4
2	2 3 5
3	1 2 3 5 2
4	252

110

Sequences (minsupp = 50 %)

120

Sequence	Support
1	50 %
2	75 %
3	75 %
5	75 %
1 3	50 %
2 2	50 %
2 3	50 %
2 5	75 %
5 2	50 %
3 5	50 %
2 5 2	50 %
2 3 5	50 %

Sequences after re-mapping (minsupp = 50 %)

130

Sequence	Support
index.htm	50 %
about.html	75 %
product.html	75 %
order.html	75 %
index.html product.html	50 %
about.html about.html	50 %
about.html product.html	50 %
about.html order.html	75 %
order.html about.html	50 %
product.html order.html	50 %
about.html order.html about.html	50 %
about.html product.html order.html	50 %

Fig. 1

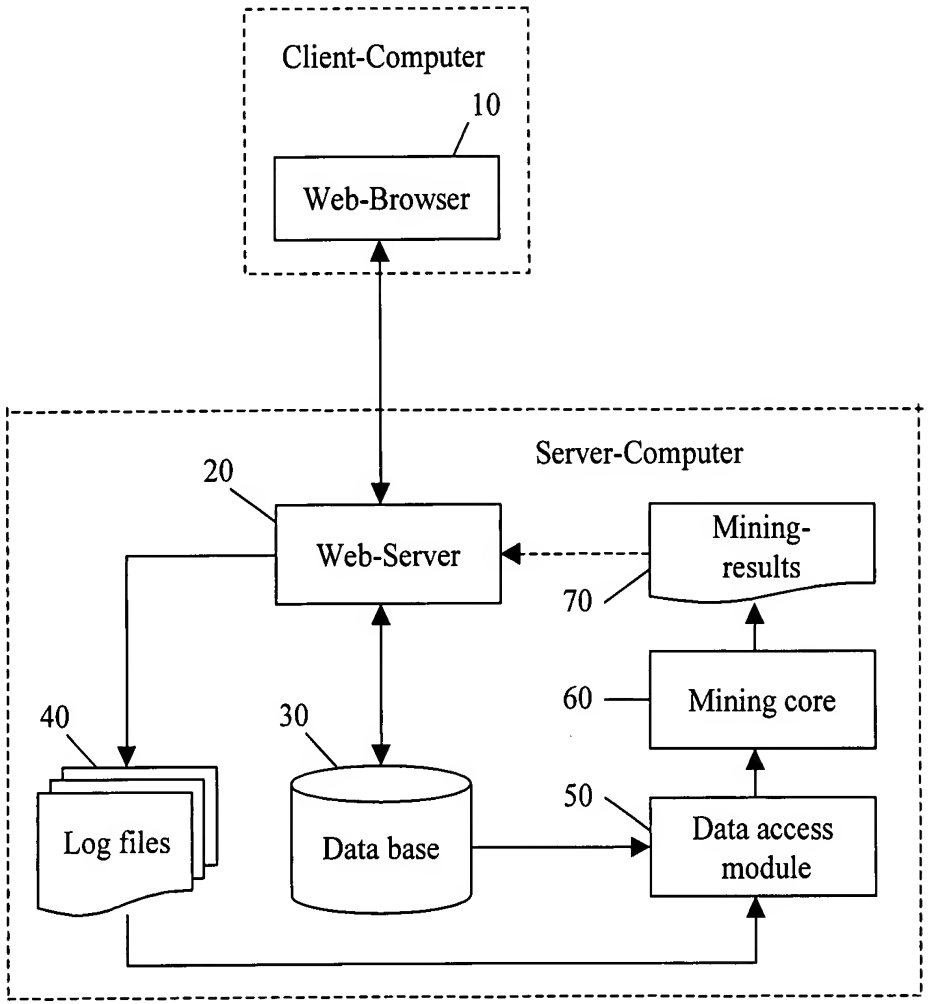
Client-Computer
10
Web-Browser
Server-Computer
20
Web-Server
Mining-
results
70
60
Mining core
40
30
Log files
Data base
50
Data access
module

Fig. 2

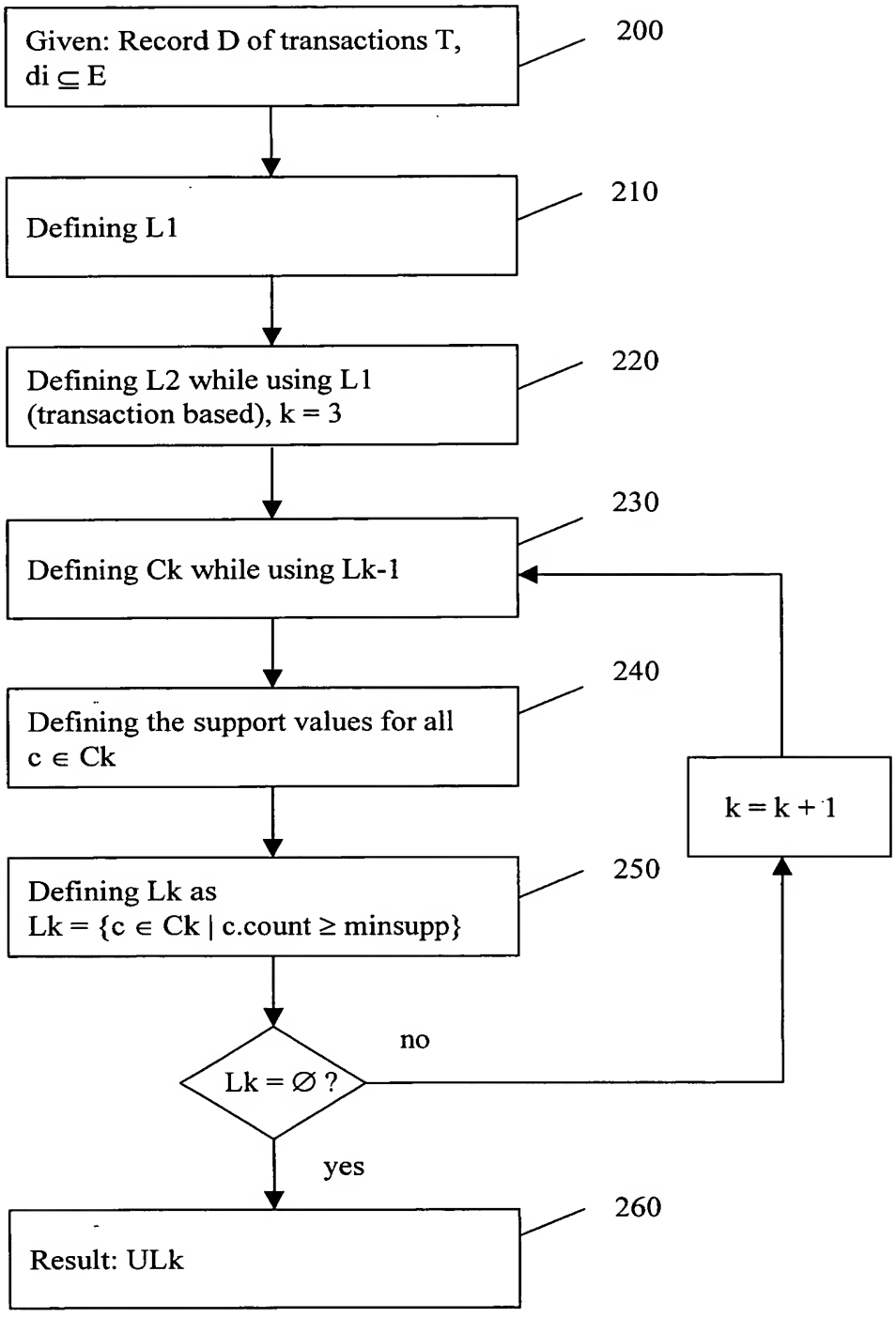
Given: Record D of transactions T, di ⊆ E
200
Defining L1
210
Defining L2 while using L1 (transaction based), k = 3
220
Defining Ck while using Lk-1
230
Defining the support values for all c ∈ Ck
240
Defining Lk as Lk = {c ∈ Ck | c.count ≥ minsupp}
250
Lk = ∅ ?
no
yes
k = k + 1
Result: ULk
260

Fig. 3

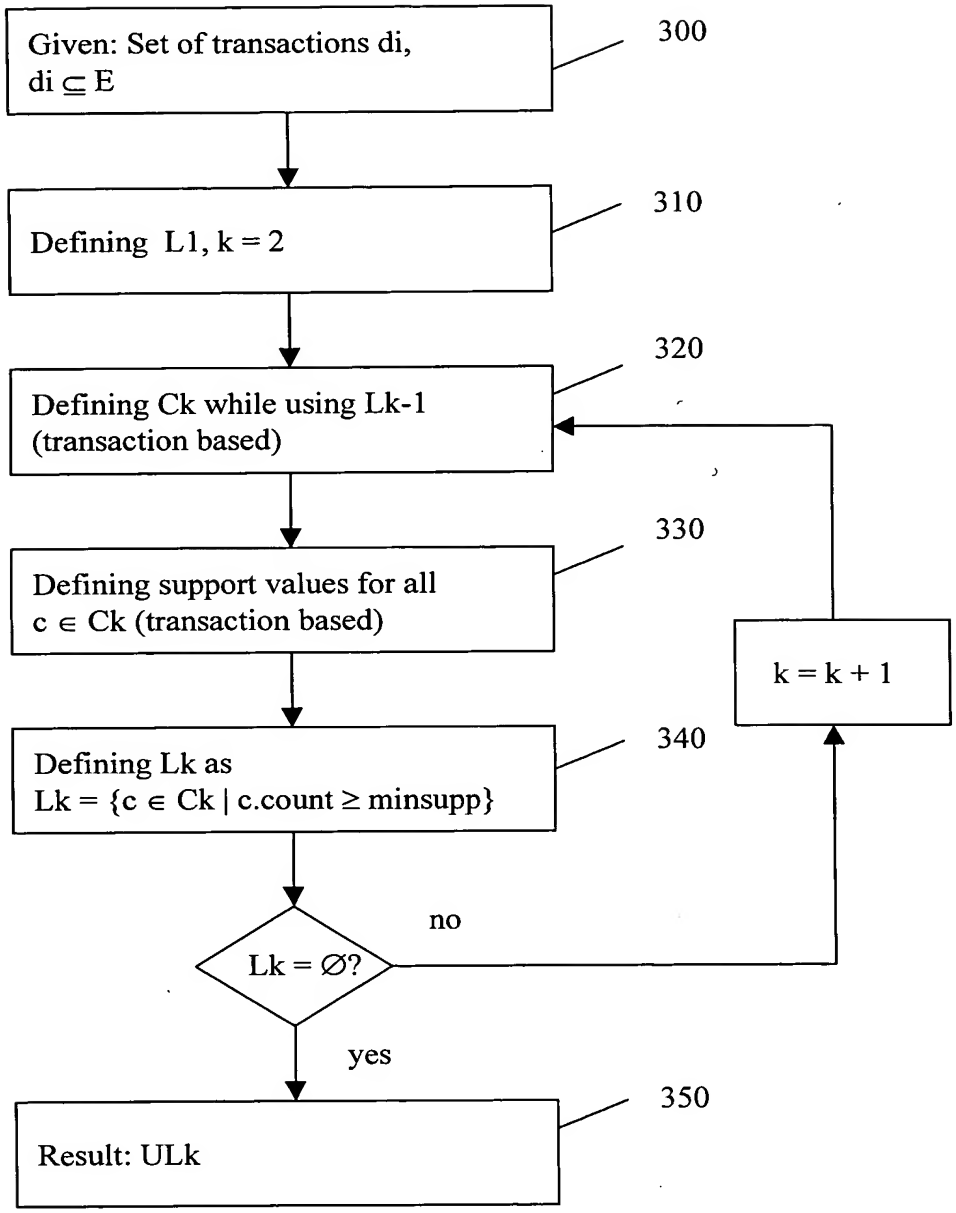
Given: Set of transactions di,
di ⊆ E
300
Defining L1, k = 2
310
Defining Ck while using Lk-1
(transaction based)
320
Defining support values for all
c ∈ Ck (transaction based)
330
Defining Lk as
Lk = {c ∈ Ck | c.count ≥ minsupp}
340
Lk = ∅?
no
yes
k = k + 1
Result: ULk
350

Fig. 4

BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	102 45 859.6
Anmeldetag:	30. September 2002
Anmelder/Inhaber:	Prudential Systems Software GmbH, Chemnitz/DE
Bezeichnung:	Verfahren zur Extraktion von Sequenzen in großen relationalen Tabellen
IPC:	G 06 F 17/60

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 13. Oktober 2003
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Faust

A 9161
02/00
EDV-L

Prudential Systems Software GmbH
P60074

Verfahren zur Extraktion von Sequenzen in großen relationalen Tabellen

Die Erfindung liegt auf dem Gebiet des „Data Mining".

Stand der Technik

Sequenzanalysen können dem Bereich des Data Mining zugeordnet werden und extrahieren Sequenzen von Elementen aus transaktionalen Daten. In dem Maße, in dem transaktionale Daten zunehmend elektronisch erfasst werden, gewinnen Sequenzanalysen an Bedeutung.

Ein typisches Beispiel für Sequenzanalysen stellt die Clickstreamanalyse dar, in welcher die typischen Nutzerpfade von Nutzern von Websites berechnet werden (siehe ebenfalls Abschnitt 8). Weitere Beispiele von Sequenzanalysen sind die Textanalyse zur Extraktion typischer Wortfolgen in elektronisch gespeicherten Dokumenten sowie die Warenkorbanalyse (unter Berücksichtigung der Produktreihenfolge) im Handel zur Bestimmung typischer Produktkettenkäufe. Vielfältige Anwendungen der Sequenzanalyse finden sich ferner in der Chemie sowie in der Genetik.

Derzeit existieren eine Vielzahl spezialisierter Verfahren zur Sequenzanalyse, insbesondere im Bereich der Genetik, jedoch sind nur wenige universelle verfügbar. Im einfachsten Fall werden hierbei alle Varianten möglicher Sequenzen auf deren Häufigkeit untersucht, doch ist dies aus Gründen der Rechenkapazität nur für kleine Datenmengen möglich. Eine Alternative sind Sequenzanalysealgorithmen, welche auf Suchbäumen basieren, beispielsweise der Algorithmus *Capri* der Firma Lumio (URL **http://www.spss.com/PDFs/CP2SPCZ-1201.pdf**). Doch ist auch deren Geschwindigkeit zur Analyse großer Transaktionsdaten unzureichend. Eine schnellere Alternative stellen Verfahren dar, welche auf Ideen der klassischen Warenkorbanalyse beruhen und bereits für die sequentielle Warenkorbanalyse genutzt wurden (Agrawal R., Srikant R. Mining Sequential Patterms. IBM Almaden Research Center; 650 Harry Road, San Jose).

Die Erfindung

Die im Rahmen der vorliegenden Anmeldung beschriebene Erfindung adaptiert die Ideen der klassischen Warenkorbanalyse konsequent für die Sequenzanalyse und entwickelt sie für das hier beschriebene Verfahren der schnellen Sequenzanalyse weiter.

Die Erfindung wird im folgenden detailliert beschrieben.

(1) Problemstellung

Es sei $I = \{i_1, i_2, \ldots, i_n\}$ ein Satz von Literalen, welche wir als Elemente bezeichnen werden. Wir bezeichnen als Sequenz $<s_0, s_1, \ldots, s_k>$ eine geordnete Liste von Elementen. Es sei D ein Satz von Transaktionen, wobei jede Transaktion T eine Sequenz von Elementen aus I ist, so daß gilt: $T \subseteq I$. Jeder Transaktion wird ein eindeutiger Identifikator zugeordnet, gekennzeichnet als TID.

Die Sequenz $<a_0, a_1, \ldots, a_n>$ *ist enthalten (Teilsequenz)* in einer anderen Sequenz $<b_0, b_1, \ldots, b_m>$, wenn natürliche nichtnegative Zahlen $i_1 < i_2 < \ldots < i_n$ existieren, so daß

$$a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_n = b_{i_n}.$$

Wir definieren, dass eine Sequenz A eine *Stützung (support)* s% auf der Menge der Transaktionen D besitzt, wenn sie in s% aller Transaktionen der Menge D enthalten ist. Wir bezeichnen eine Sequenz als *groß (large)*, wenn gilt, dass ihr Support nicht kleiner als ein vorgegebener Minimalwert ist. Eine Sequenz der Länge k bezeichnen wir als k-Sequenz. Im weiteren werden wir die Menge aller k-Sequenzen als L_k bezeichnen. Die Menge aller k-Sequenzen, welche potenziell groß sein können, werden wir als C_k bezeichnen.

Die Menge I kann eindeutig auf eine Untermenge natürlicher Zahlen abgebildet werden, das heißt, jedem Element aus I kann ein eindeutiger natürlicher Identifikator zugeordnet werden.

Unter Benutzung der oben eingeführten Begriffe kann die Aufgabe folgendermaßen formuliert werden. Auf Basis der vorgegebenen Mengen I und D sollen alle Sequenzen gefunden werden, deren Supportwert kleiner oder gleich einem vorgegebenen Minimalwert ist, das heißt, wir suchen die Menge

$$L = \bigcup_k L_k$$

(2) Lösung

1. Grundkonzeption

Der erste Ansatz zur Lösung der vorliegenden Aufgabe besteht in der Auflistung aller möglichen Sequenzen aus Elementen der Menge *I* und dem Auszählen der sie enthaltenden Transaktionen. Wir finden die Anzahl aller möglichen Sequenzen aus Elementen der Menge *I*, der Länge von 1 bis *k*. Die Mächtigkeit der Menge *I* sein *n*. Die Anzahl der Sequenzen der Länge 1 ist gleich der Anzahl der Elemente der Menge *I*, also *n*. Die Anzahl aller möglichen Sequenzen der Länge *m* aus *n* Elementen ist n^m. Dann ist die Anzahl aller Sequenzen *N* gleich der Summe der geometrischen Reihe zur Basis *n*:

$$N = \sum_{m=1}^{k} n^m = \frac{n^{k+1} - n}{n-1}.$$

Bereits für $n = 100$ und $k = 10$ erhalten wir $N = \frac{100^{11} - 100}{100 - 1} \approx 10^{20}$ verschiedene Sequenzen.

Es ist klar, dass ein solcher Ansatz zur Lösung der Aufgabe unakzeptabel ist. Für reale Daten sind in allen Transaktionen der Menge *D* deutlich weniger Sequenzen enthalten.

Weit logischer und schneller ist der Test aller Sequenzen, welche in jeder Transaktion enthalten sind und die Bestimmung des Supportwertes für jede von ihnen. Angenommen, die Länge der Transaktion t_0 ist gleich *n*. Dann enthält die Transaktion t_0 die Anzahl aller Sequenzen der Länge *m*, welche gleich der Menge der *m* Kombinationen der Elemente über *n* ist. Dann ist die gesamte Anzahl der Sequenzen *N*, welche in einer Transaktion der Länge *n* enthalten sind, gleich:

$$N = \sum_{i=1}^{n} \frac{n!}{i! \cdot (n-i)!} = 2^n - 1 .$$

Eine Transaktion der Länge 10 enthält 1023 Sequenzen, eine Transaktion der Länge 20 enthält 1048575 Sequenzen der Längen von 1 bis 10 und entsprechend 20. Obwohl die

angeführten Zahlen den Worst Case beschreiben, kann die Anzahl der möglichen Sequenzen, welche in einer Transaktion enthalten sind, dennoch sehr groß werden.

Jede dieser Sequenzen muß auf ihre minimale Supportbedingung getestet, das heißt, es muß die Summe der sie enthaltenen Transaktionen gebildet werden. Es ist klar, dass dieser Prozeß ebenfalls sehr zeitintensiv ist. In den hier beschriebenen Verfahren wird deshalb ein Ansatz verwendet, welcher keine überflüssigen Tests benötigt.

Es werden zwei Variante der Extraktion großer Sequenzen beschrieben. Beide nutzen einen gemeinsamen Satz grundlegender Ideen, aber unterscheiden sich in der Summation der Supportwerte der Sequenzen.

Die erste Idee geht auf einen Ansatz von IBM (Agrawal R., Strikant R. *Fast Algorithms for Mining Association Rules*. IBM Almaden Research Center; 650 Harry Road, San Jose) zurück. Jedoch wurde der darin beschriebene Ansatz zur effizienten Realisierung der klassischen Warenkorbanalyse genutzt, in der die Reihenfolge der Elemente in den Transaktionen unberücksichtigt bleibt. Einer der Grundansätze der vorliegenden Beschreibung ist die Adaption dieser Ideen, welche ursprünglich nicht zur Lösung von Aufgaben unter Berücksichtigung der Reihenfolge der Elemente gedacht waren.

Die hier beschriebenen Verfahren gehören zur Klasse der Iterationsverfahren. In jedem Arbeitsschritt der Verfahren werden alle großen Sequenzen einer gegebenen Länge bestimmt. Die Arbeit der Verfahren dauert solange, bis alle großen Sequenzen maximaler Länge gefunden wurden, das heißt, bis im vorliegenden Arbeitsschritt keine weiteren großen Sequenzen gefunden werden. Ein anderes Kriterium zur Terminierung des Verfahrens kann das Erreichen einer maximalen Länge der Sequenzen sein, welche vom Nutzer vorgegeben worden ist.

In beiden Varianten des Verfahrens werden die großen Sequenzen der Länge k benutzt, um die Menge neuer potentieller großer Sequenzen der Länge k+1 zu konstruieren. Wir bezeichnen eine Sequenz als *Kandidat*, wenn diese aus großen Sequenzen der um Eins verminderten Länge generiert worden ist und möglicherweise ebenfalls eine große Sequenz darstellt. Auf diese Weise bildet die Menge der Kandidaten der Länge k die Menge C_k. Offensichtlich gilt: $C_k \supseteq L_k$.

2. Die Methode zur Gerierung der Kandidatenmenge

Beide Verfahren nutzen das gleiche Prinzip der Generierung von Kandidaten, welches sich etwas von dem der Generierung der Kandidaten im oben genannten Ansatz von IBM unterscheidet. Die Funktion zur Generierung der Kandidaten nutzt als Eingangsmenge alle großen k-Sequenzen – L_k. Sie liefert im Ergebnis eine Supermenge der Mengen aller großen $(k+1)$-Sequenzen – C_{k+1}. Unter Nutzung einer SQL-artigen Syntax kann das Verfahren der Generierung der Kandidatenmenge folgendermaßen aufgeschrieben werden:

INSERT INTO C_k
SELECT p.item$_1$,p.item$_2$,…,p.item$_k$,q.item$_k$
FROM p,q
WHERE p.item$_2$= q.item$_1$, p.item$_3$= q.item$_2$, …, p.item$_k$ = q.item$_{k-1}$

Die Arbeitsweise der Generierung der Kandidaten kann anhand folgenden Beispiels veranschaulicht werden.

Angenommen, in einem Schritt der Arbeit des Verfahrens sind folgende große Sequenzen der Länge 4 gefunden worden:

1. 1 3 2 6
2. 1 4 1 2
3. 2 1 3 2
4. 2 2 2 2
5. 3 1 2 1
6. 3 2 6 2
7. 4 1 2 1

Mittels Durchgehen aller möglichen Paare großer 4-Sequenzen, beginnend mit dem ersten, und unter Einschluß auch aller aus der gleichen Sequenz bestehenden, erhält man folgende Kandidatenmenge:

1. 1 3 2 6 2 – Ergebnis der Vereinigung von 1 und 6
2. 1 4 1 2 1 – Ergebnis der Vereinigung von 2 und 7
3. 2 1 3 2 6 – Ergebnis der Vereinigung von 3 und 1

4. 2 2 2 2 2 – Ergebnis der Vereinigung von 4 und 4

Die Sequenzen 5, 6 und 7 lieferten kein einziges Paar, in welchem sie die ersten der „Eltern" wären; die Sequenz 5 war überhaupt nicht an der Generierung von Kandidaten beteiligt; die Sequenz 4 wurde mit sich selbst vereinigt.

Im Ergebnis der Arbeit des Verfahrens kann eine Situation entstehen, in der mittels dieser Methode keine einzige Kandidaten-Sequenz generiert wird. Das bedeutet, dass es nicht möglich ist, auch nur eine einzige Sequenz des nächsten, k+1-ten, Levels zu finden. Folglich terminiert das Verfahren in diesem Schritt.

Im dargelegten Beispiel wurden zur Konstruktion der Kandidatenmenge alle möglichen Paare großer Sequenzen des vorhergehenden Levels herangezogen. Während der Arbeit des Verfahrens auf realen Daten kann es leicht passieren, in einem gewissen Schritt eine Menge aller großen Sequenzen zu finden, welche aus Tausenden oder Zehntausenden Sequenzen besteht. Der Prozeß der vollständigen Durchsicht aller Paare besitzt eine quadratische Komplexität. Daher müsste man auf realen Daten Millionen oder Hunderte von Millionen Operationen zum Vergleich von Sequenzen ausführen. Das kann zu ernsthaftem Zeitaufwand für den Prozeß der Kandidatengenerierung führen.

Daher wurde eine Datenstruktur erarbeitet, welche es erlaubt, die Komplexität der Durchsicht der Paare großer Sequenzen zu verringern. Gemäß dieser Idee ist es notwendig, für jede große k-Sequenz die erste und die letzte große (k+1)-Sequenz zu speichern, welche mit der betrachteten k-Sequenz beginnen. Natürlich ist es dafür notwendig, daß der Satz der k-Sequenzen von den kleinen k-Sequenzen zu den großen Sequenzen sortiert wird. Die k-Sequenz A wird als kleiner als die k-Sequenz B definiert, wenn das erste verschiedene Element in der Sequenz A kleiner als in der Sequenz B ist.

Somit besitzen wir in einem Schritt des Verfahrens L_k – die Menge der großen k-Sequenzen des vorangegangenen Schrittes, C_{k+1} – die Menge der Kandidaten. Weiterhin wird für jede k-Sequenz aus L_k die Nummer des zweiten „Elternteils" gespeichert, welcher diese Sequenz generiert hat. Unter Benutzung dieser Nummer kann für jede k-Sequenz gleich das Spektrum der Sequenzen aus L_k bestimmt werden, mit welchen diese vereinigt werden kann. Diese Datenstruktur wird in beiden Varianten des beschriebenen Verfahrens genutzt. Zur

Speicherung der benötigten Nummern für die Sequenzen aus L_{k-1} wird ein dynamisches Array der Dimension $|L_{k-1}|\times 2$ genutzt.

Im k-ten Schritt liegt die Menge der großen Sequenzen L_k vor, mit jeder von denen die Nummer derjenigen Sequenz aus L_{k-1} verbunden ist, welche den zweiten „Elternteil" bildet. Im Ergebnis der Arbeit erhält man im k-ten Schritt aus C_{k+1} die Menge L_{k+1}, für deren Sequenzen die Nummer ihres zweiten Elternteils gespeichert werden, und es wird eine neue Struktur aus den Nummern der Sequenzen aus L_k geformt.

Im oben angeführten Beispiel wurde die Sequenz <2 1 3 2> im vorhergehenden Schritt aus den Sequenzen <2 1 3> und <1 3 2> geformt. Angenommen, die Sequenz <1 3 2> stand in der Nummer 3 der Liste der Sequenzen L_3. Dann müssen in der beschriebenen Struktur für die Sequenz der Nummer 3 die Nummern 1 und 1 gespeichert werden – entsprechend der ersten und letzten Sequenz aus L_4, welche mit <1 3 2> beginnen. Für die neue Sequenz <2 1 3 2 6> wird, falls sie sich als groß erwiesen hat, die Nummer ihres zweiten Elternteils gespeichert – 1, für die Sequenz <2 1 3 2> die Nummer 3 – die Nummer der ersten neuen erhaltenen 5-Sequenz.

Wenn für eine Sequenz aus L_{k-1} kein Nachfahre existiert, wird dieser Fakt in der Struktur der Nummern über eine ungültige Nummer gespeichert – beispielsweise -1.

(3) Beschreibung der ersten Variante des Verfahrens

Der Prozeß der Arbeit des ersten Verfahrens kann in mehrere Etappen aufgegliedert werden:

1. Suche aller großen 1-Sequenzen – der verschiedenen Elemente aus $I - L_1$.

2. Suche aller großen 2-Sequenzen – L_2 auf Grundlage von L_1. Gleichzeitig wird für jede gefundene große Sequenz die Liste der Transaktionsnummern, welche diese enthalten, abgespeichert.

3. Suche der großen k-Sequenzen auf Grundlage von L_{k-1}. Die Etappe 3 wird solange wiederholt, bis $L_k = \varnothing$ erhalten wurde.

Die Beschreibung des ersten Verfahrens im Pseudocode ist in Abbildung 1 dargestellt.

```
L1 = {large 1-sequences};
C2 = candidate-gen(L1);                    // 2-Kandidaten
forall transactions t ∈ D do begin
        Ct = subset(C2,t);        // Kandidaten, welche in t enthalten sind
        forall candidates c ∈ Ct do
                c.count++;
end
L2 = {c ∈ C2| c.count ≥ minsupp};
for(k=3; Lk-1 ≠ ∅; k++) do begin
        Ck = candidate-gen(Lk-1);
        forall candidates c ∈ Ck do begin
                Tc = subset(c,D);         // Transaktionen, welche c enthalten
                c.support = |Tc|;
        end
        Lk = {c ∈ Ck | c.count ≥ minsupp};
end
answer = ULk;
```

Abbildung 1: Erste Variante des Verfahrens im Pseudocode.

In der ersten Variante des Verfahrens werden zwei Ansätze zur Auszählung der Supportwerte der Kandidaten-Sequenzen kombiniert. Zum Auffinden der Menge L_2 wird der sequentielle Durchgang durch die Menge der Eingangstransaktionen genutzt. Für jede Transaktion werden die darin enthaltenen Kandidaten bestimmt, und für jeden Kandidaten wird der Zähler des Supportwertes um Eins erhöht.

In der ersten Variante des Verfahrens wurde in dieser Etappe der gleiche Ansatz gewählt wie in der nächsten Etappe: Für jeden Kandidaten wird die Liste der Transaktionen bestimmt, welche diesen Kandidaten enthalten können. Die Liste der Transaktionen ergibt sich als Schnittmenge über den gespeicherten Transaktionslisten der Eltern des betrachteten Kandidaten. Offensichtlich kann die Kandidaten-Sequenz nur in den Transaktionen enthalten sein, in denen gleichzeitig beide Elternteile dieses Kandidaten gespeichert sind. Die Liste der Transaktionen des Kandidaten enthält die Nummern der Transaktionen und unter Nutzung der Schnittmengenbildung werden dahin nur diejenigen Nummern hinzugefügt, welche in den

Listen der Transaktionen seiner beiden Elternteile enthalten sind. Die derart erhaltene Liste erlaubt eine substantielle Reduktion der Anzahl der Transaktionen, welche auf das Vorhandensein eines bestimmten Kandidaten getestet werden müssen, besonders in den letzten Schritten des Verfahrens.

Jedoch setzt der Ansatz der Transaktionslisten den Durchlauf durch die Gesamtmenge der Kandidaten voraus. In der zweiten Etappe des Verfahrens, bei der Auszählung der Menge L_2, sind nur die Elemente der Menge L_1 bekannt. Ein beliebiges Paar großer 1-Sequenzen bildet einen Kandidaten der Länge 2. Während der Arbeit des Verfahrens mit realen Daten kann die Mächtigkeit der Menge L_1 einige zehntausend 1-Sequenzen erreichen. Dann beträgt die Anzahl der Kandidaten der Länge 2 einige hundert Millionen. Eine vollständige Durchsicht aller möglichen Kandidaten würde einen unvertretbar hohen Zeitaufwand erfordern. Es ist sinnvoller, in diesem Fall den Ansatz zu wählen, welcher vollständig in der zweiten Variante des Verfahrens benutzt wurde.

Gemäß der grundlegenden Idee des zweiten Verfahrens, wird für jeden Kandidaten ein Zähler seines Supportwertes eingeführt. Danach erfolgt der sequentielle Durchgang durch die Eingangsmenge der Transaktionen. Für jede Transaktion werden die in ihr enthalten Kandidaten bestimmt und ihre Zähler des Supports werden um Eins erhöht. In der zweiten Variante des Verfahrens wird ein komplexeres Schema zur Bestimmung der Kandidaten genutzt, welche in einer konkreten Transaktion enthalten sind, aber zur Bestimmung der Menge L_2 kann eine einfachere Methode verwendet werden. Zur Bestimmung der Nummern aller Kandidaten sind zwei Durchläufe durch die Transaktionen notwendig, von denen der eine in den anderen eingebettet ist. Im äußeren Zyklus werden nacheinander die großen Elemente gewählt, welche in den Transaktionen enthalten sind; die nicht-großen Elemente werden ausgelassen. Der innere Zyklus beginnt mit dem Element, welches auf das in dem äußeren Zyklus ausgewählte Element folgt. Jedes ausgewählte Paar großer Elemente der Menge I stellt einen Kandidaten der Länge 2 dar und folglich ist es notwendig, den entsprechenden Zähler des Supportwertes des Kandidaten zu erhöhen. Die Nummer des Zählers wird ausgehend aus den Indexen der ausgewählten Elemente in der Liste der großen Elemente und der Gesamtzahl der gefundenen großen Elemente berechnet. Seien a und b – die entsprechend gewählten ersten und zweiten Elemente eines Kandidaten und N – die Anzahl der in der vorhergehenden Etappe des Verfahrens gefundenen 1-Sequenzen – der großen Elemente aus L_1. Dann ergibt sich die Nummer des entsprechenden Kandidaten gemäß folgender Formel: $n_c = Na + b$.

Zum Ausschließen der wiederholten Erhöhung des Zählers des Supports eines speziellen Kandidaten bei der Behandlung ein und derselben Transaktion ist es notwendig, eine Methode zur Markierung des Faktes der Erhöhung des Zählers einer konkreten Transaktion vorzusehen. Die einfachste Methode besteht darin, für jeden Kandidaten außer seinem Zähler des Supports zusätzlich die Nummer der letzten Transaktion zu speichern, welche den Zähler des betrachteten Kandidaten inkrementiert hat. Da die Transaktionen aus der Eingangsmenge nacheinander behandelt werden, kann über den Inhalt des Feldes der Nummer der letzten Transaktion festgestellt werden, ob der Zähler seit Beginn der Behandlung der betrachteten konkreten Transaktion erhöht wurde.

Der Vorteil der beschriebenen Vorgehensweise zur Bestimmung der 2-Sequenzen L_2 ist die relative Einfachheit der Berechnungen, da nur diejenigen Kandidaten behandelt werden, welche tatsächlich in den Eingangstransaktionen enthalten sind. Allerdings besitzt diese Methode auch einen gravierenden Nachteil. Für jeden Kandidaten müssen zwei mit ihm verbundene Werte gespeichert werden: der Zähler des Supportwertes und die Nummer der letzten, den Zähler inkrementierenden, Transaktion. Die Nutzung eines 32-bit Wertes für jedes Feld, welches theoretisch die Bearbeitung von maximal 2147483647 Transaktionen erlaubt, erfordert 8 Byte pro Kandidat. In der Etappe der Bestimmung der Menge L_2 können Hunderte Millionen verschiedener Kandidaten existieren. Folglich benötigt diese Etappe der Arbeit des Verfahrens einige hundert Millionen Byte RAM-Speicher, was durchaus eine ernsthafte Einschränkung darstellt. Die quadratische Abhängigkeit des benötigten Speichers von der Anzahl der in der vorhergehenden Etappe gefundenen großen 1-Sequenzen stellt eine ernsthafte Einschränkung dar.

Um diese Beschränkung zu beseitigen, wurde in der endgültigen Form des ersten Verfahrens ein Schema des Mehrfachdurchlaufes der Bearbeitung der Transaktionsmenge im Verlauf der Konstruktion der Menge L_2 realisiert. Während eines vollständigen Durchlaufes durch die Eingangsmenge der Transaktionen werden nur diejenigen Paare behandelt, deren erstes Element sich in einem bestimmten Wertebereich befindet. Auf diese Weise kann der Speicherplatz, welcher für die zweite Etappe des Verfahrens benötigt wird, unter Berücksichtigung des im Rechner verfügbaren Speicherplatzes beschränkt werden. Selbstverständlich werden mit stärkeren Speicherbeschränkungen umso mehr Durchläufe erforderlich und das Verfahren wird in der zweiten Etappe entsprechend langsamer. Daher stellt die beschriebene Methode der Bestimmung der 2-Sequenzen L_2 einen Kompromiß

zwischen benötigtem Speicherplatz und Bearbeitungsgeschwindigkeit dar. Diese Tatsache wird auch durch das experimentelle Laufzeitverhalten des Verfahrens bestätigt.

(4) Details der Realisierung der ersten Variante

In der ersten Etappe der Arbeit des Verfahrens müssen alle Elemente der Menge *I* gefunden werden, welche in den Eingangstransaktionen nicht weniger als eine vorgegebene Anzahl auftauchen. Da die Nummern und Anzahlen der verschiedenen Elemente aus *I* anfänglich nicht bekannt sind, ist es notwendig, eine Methode zur Speicherung des eigentlichen Elementes sowie der Anzahl der Transaktionen, in der dieses enthalten ist, vorzusehen. In der Realisierung beider Varianten des Verfahrens erfolgt ein Durchlauf durch alle Eingangstransaktionen. Zur Speicherung der Elemente und ihrer Zähler wird eine Standardklasse (z.B. aus Java) herangezogen – die *Hashtable*. Diese Klasse erlaubt die Speicherung des Paares <Schlüssel, Wert>, welche zur schnellen Suche der Werte den Mechanismus des Hash-Tabelle bezügliche der Schlüsselwerte benutzt. In der Realisierung des Verfahrens wird als Schlüssel ein Objekt der Klasse (z.B. aus Java) *Integer* (Wrapper-Klasse für Integer-Werte) benutzt, welche mit dem Wert des gewählten Elementes initialisiert wird, und als Wert – ein anderes Objekt der Klasse *Integer*, welches mit dem Wert seines Zählers initialisiert wird. Bei der Auswahl eines weiteren Elementes einer Transaktion erfolgt ein Zugriff zum Objekt der Klasse *Hashtable*, um den Wert über den Schlüssel zu erhalten, welcher gleich dem gewählten Element ist. Wenn ein solcher Wert in der Hash-Tabelle gefunden wurde, so wird es erneut, um Eins erhöht, zurückgeschrieben. Wenn kein solcher Wert für den Schlüssel gefunden wurde, so wird in die Hash-Tabelle das Paar eingetragen, welches als Wert den Anfangswert des Zählers unter Berücksichtigung des bereits gewählten Elementes enthält – 1. Zur Vermeidung wiederholter Inkrementierungen des Zählers einer Transaktion wird in der Realisierung des Verfahrens eine andere Klasse benutzt (z.B. aus Java) – *Vector*, in der alle bereits von einer Transaktion bearbeiteten Elemente gespeichert werden. Bevor es den Wert in der Hash-Tabelle sucht, versucht das Verfahren das Element in der Instanz der Klasse *Vector* zu finden. Wenn ein solches Element gefunden wurde, so bedeutet dies, dass es bereits in dieser Transaktion bearbeitet wurde und es findet ein Übergang zum nächsten Element der Transaktion statt. Natürlich wird der Vektor der Werte beim Übergang des Verfahrens zur nächsten Transaktion gesäubert. Nach dem Durchgang durch die Transaktionsmenge werden nur jene Elemente selektiert, deren Zählerwerte nicht kleiner als der Minimalwert sind. Die gefundenen großen Elemente werden im Array abgespeichert und werden sortiert, um im weiteren Verfahren die Binärsuche der Elemente

anwenden und auf die Folge der ganzen positiven Zahlen 0, ..., N-1 abbilden zu können, wobei N die Anzahl der gefundenen großen Elemente ist. Aus den großen Elementen wird die Menge L_1 der großen 1-Sequenzen formiert.

Zur Speicherung der Liste der Nummern der Transaktionen wird in der Realisierung des Verfahrens die Klasse *TransactionList* benutzt. Sie erlaubt die Speicherung der Nummern der Transaktionen in einem Array, das Hinzufügen einer Nummer an das Ende der Liste, wobei vorausgesetzt wird, dass die Nummern in aufsteigender Reihenfolge hinzugefügt werden. Diese Voraussetzung ist unabdingbar für die korrekte Arbeit der Methode der Klasse *TransacionList*, welche die Schnittmengenbildung beider Transaktionslisten realisiert. Die Anfangslisten der Transaktionen werden im Verlauf des sequentiellen Durchlaufes durch die Menge der Eingangstransaktionen in der Etappe der Konstruktion von L_2 gebildet, hierbei werden in natürlicher Weise die Bedingungen der Sortierordnung sichergestellt. Im weiteren entstehen die neuen Listen während der Ausführung der Schnittmengenoperation. Die Listen der Nummern werden in einem Array gespeichert, aber da die Länge des Arrays in der Etappe der dynamischen Speicherallokation bestimmt werden, ist es notwendig, im erforderlichen Maße zusätzliche Felder des Arrays bereitzustellen, indem seine Länge erhöht wird. Während der Erzeugung einer Instanz der Klasse *TransactionList* wird ein Array mit einer Anfangsgröße angelegt und schrittweise gefüllt. Bei Erreichen der Maximalanzahl der Nummern wird der Speicher erneut bereitgestellt. Die maximale Speicherkapazität des Arrays wird beispielsweise um das Zehnfache erhöht, es wird ein neues Array der gleichen maximalen Länge bereitgestellt und darin wird der Inhalt des vorhergehenden Arrays kopiert. Das alte Array kann als überflüssig markiert werden (z.B. für die Virtual Machine in Java).

In der dritten Etappe der Arbeit des Verfahrens werden alle im vorhergehenden Schritt gespeicherten großen k-Sequenzen durchgegangen. Über die gespeicherte Nummer des zweiten Elternteils der Sequenz werden, unter Nutzung der oben beschriebenen Datenstruktur, die Nummern der ersten und letzten großen k-Sequenzen gefunden, mit denen die gefundene Sequenz vereinigt werden kann. Weiter wird die Schnittbildungsoperation über den Listen der Transaktionen beider Sequenzen ausgeführt. Wenn die Länge der resultierenden Liste kleiner als der notwendige minimale Supportwert ist, so wird der betrachtete Kandidat sofort verworfen, sogar ohne Operation eines unmittelbaren Kandidaten und der Auszählung der Anzahl der Transaktionen, welche diesen enthalten. Andernfalls erfolgt die Erzeugung der Sequenz der Elemente des Kandidaten selbst und die Überprüfung jeder Transaktion aus der Schnittbildung erhaltenen Liste auf den Inhalt der in ihr enthaltenen Kandidaten-Sequenzen.

Wenn in einer Transaktion eine entsprechende Sequenz gefunden wurde, so wird die Nummer dieser Transaktion gespeichert. Auf diese Weise liegt nach der Bearbeitung eines Kandidaten eine Liste der Transaktionen vor, welche den Kandidaten enthalten. Wenn die Anzahl solcher Transaktionen nicht kleiner der Mindestanzahl ist, so wird dieser Kandidat zusammen mit der Nummer seines zweiten Elternteils gespeichert.

Die oben beschriebenen Aktionen werden für jeden Kandidaten wiederholt. Wenn im Ergebnis keine einzige große Sequenz gefunden wurde, so wird die Arbeit der dritten Etappe des Verfahrens beendet und die Ergebnisse werden über das Interface ausgegeben. Es kann zwei Gründe geben, warum im Verlauf eines Schrittes keine einzige große Sequenz gefunden wurde. In der Menge L_k kann keine Sequenz vorliegen, welche einen Kandidaten mit einer anderen Sequenz bilden könnte, dass heißt, in einem Schritt existiert kein einziger Kandidat. Ein anderer Grund kann die Variante sein, in der im Ergebnis kein einziger Kandidat eine große Sequenz darstellt.

Fassen wir zusammen. Die erste Variante des Verfahrens erweist sich als schnell auf großen Datenmengen und weist einen sparsamen Speicherplatzbedarf auf. Schwer vorhersehbar sind der Aufwand bezüglich Speicherplatz und Rechenzeit hinsichtlich der Benutzung von Transaktionslisten, aber empirische Berechnungen und experimentelle Resultate zeigen, dass auf realen Daten ihre Größe eine Tendenz zur starken Verringerung aufweist, da mit Erhöhung der Länge der bearbeiteten Sequenzen diese in einer zunehmend geringeren Anzahl von Transaktionen enthalten sind und sich folglich die Länge der Listen verringert wie auch die zu ihrer Bearbeitung benötigte Zeit.

Folglich muß für die Arbeit der ersten Variante des Verfahrens die Eingangsmenge der Transaktionen in Form einer Liste von Sequenzen ganzer Zahlen eingegeben werden, der minimale Supportwert als Floatwert und ferner der verfügbare Speicherplatz für die zweite Etappe des Verfahrens.

(5) Beschreibung des zweiten Variante des Verfahrens

Die zweite Variante des Verfahrens wurde zum Zweck des Vergleichs mit der ersten Variante des Verfahrens bezüglich Geschwindigkeit und Speicherplatz erarbeitet und realisiert. In Abbildung 2 wurde im Pseudocode das Prinzip der Arbeitsweise der zweiten Variante des Verfahrens dargestellt.

```
L1 = {large 1-sequences};
for (k=2; Lk-2 ≠ ∅; k++) do begin
        Ck = candidate-gen(Lk-i);        // neue Kandidaten
        forall transactions t ∈ D do begin
                Ct = subset(Ck, t);
                forall candidates c ∈ Ct do
                        c.count++;
        end
        Lk = {c ∈ Ck | c.count ≥ minsup}
end
answer = U Lk;
```

Abbildung 2: Zweite Variante des Verfahrens im Pseudocode.

Das dargestellte Verfahren ist sehr allgemein und konkretisiert nicht die Methode der Auswahl der Kandidaten, welche in einer konkreten Transaktion enthalten sind. Im einfachsten Fall kann dies die Durchsicht aller Kandidaten sein mit dem Ziel, diejenigen auszuwählen, welche in der Sequenz der Transaktion enthalten sind. Eine andere Variante ist die Durchsicht aller Sequenzen einer gegebenen Länge und die Auswahl derjenigen aus ihnen, welche Kandidaten darstellen. Es ist klar, dass beide Methoden inakzeptabel wegen deren geringer Effizienz sind; in der Realisierung der zweiten Variante wird daher eine andere Methode beschrieben.

Die zweite Variante benutzt eine Reihe gleicher Ideen wie die erste Variante. Insbesondere wird zur Berechnung der Menge L_1 der gleiche Ansatz ohne große Änderungen benutzt. Die gesamte Beschreibung der ersten Etappe der Arbeit der ersten Variante des Verfahrens, welche oben angeführt wurde, gilt auch bezüglich der zweiten Variante. Wie bereits oben erwähnt, benutzt die zweite Variante des Verfahrens zur Summierung des Supports der Kandidaten die Idee, welche in der ersten Variante nur in der zweiten Etappe zur Generierung der Menge der großen 2-Sequenzen L_2 benutzt wurde.

(6) Details der Realisierung der zweiten Variante

Um für jede Transaktion die Möglichkeit zur Bestimmung der Liste der in ihr enthaltenen Kandidaten zu erhalten, wird in der zweiten Variante des Verfahrens eine spezielle

Datenstruktur benutzt. Beginnend mit der zweiten Etappe des Verfahrens wird jede Transaktion als Liste der Nummern der im vorhergehenden Schritt des Verfahrens gefundenen *k*-Sequenzen dargestellt. Unter Durchsicht der Paare dieser *k*-Sequenzen und deren Zusammenfügung gemäß der oben beschriebenen Methode der Kandidatengenerierung kann man alle Kandidaten der Länge *k*+1 erhalten, welche in einer bestimmten Transaktion enthalten sind. Jedoch zur Trennung der Kandidaten ist die Liste der großen *k*-Sequenzen als zweidimensionales Array von Werten organisiert. Die resultierende Datenstruktur wird im nächsten Beispiel illustriert:

Angenommen, die Eingangsmenge der Transaktionen enthält folgende Transaktion:

<3 1 3 2 1>

Wir nehmen weiterhin an, dass in der zweiten Etappe der Arbeit des Verfahrens festgestellt wurde, dass die in dieser Transaktion enthaltenen Sequenzen <3 1>, <3 2>, <1 2> und <1 1> groß sind und jeweils die Nummern 5, 7, 3 und 1 besitzen. Dann wurde im Ergebnis der zweiten Etappe der Arbeit des Verfahrens folgende Datenstruktur erzeugt, welche der gegebenen Transaktion entspricht:

1. 5 7
2. 3 1
3. 7 5
4. –

Die Anzahl der Zeilen in dieser Struktur ist gleich der Anzahl der Elemente in der Transaktion während der Etappe der Konstruktion der Menge L_2. In den weiteren Etappen ist diese Menge gleich der Zahl der Zeilen in der vorhergehenden Struktur für diese Transaktion. Jede Zeile enthält die Nummern der großen Sequenzen, welche mit diesem Element beginnen oder mit dieser Zeile in der Struktur, welche im vorhergehenden Schritt erzeugt wurde. In jedem Schritt werden die Paare der Nummern der großen Sequenzen des vorhergehenden Levels betrachtet und das Verfahren durchsucht alle Paare von Nummern, welche vereinigt werden können. Der Test auf die Möglichkeit zur Verknüpfung wird auf Basis des gleichen Prinzips wie in der ersten Variante des Verfahrens ausgeführt.

Eine derartige Struktur wird für jede Transaktion gebildet und wird in jedem Schritt des Verfahrens aktualisiert. Die Erststruktur wird in der zweiten Etappe des Verfahrens erzeugt und das ist deren einziger Unterschied zur entsprechenden Etappe der ersten Variante des Verfahrens. In einem Schritt des Verfahrens werden nacheinander alle Transaktionen der Eingangsmenge behandelt, wobei die beschriebene Struktur genutzt wird. Schrittweise wird jedes Element jeder Zeile gewählt und danach werden die Elemente jeder nachfolgenden Zeile durchsucht. Wenn das derart gewählte Paar einen Kandidaten darstellen kann, so wird der Zähler seines Supports um Eins erhöht. Zur Verhinderung einer mehrfachen Inkrementierung des Zählers während der Bearbeitung einer Transaktion kann die Idee der ersten Variante des Verfahrens herangezogen werden – für jeden Kandidaten wird die Nummer der letzten inkrementierenden Transaktion gespeichert.

Auf diese Weise zeigt sich, dass in der zweiten Variante des Verfahrens zur Laufzeit gar keine reelle Generierung von Kandidaten-Sequenzen vonnöten ist, da dies nicht für die Summierung der Supportwerte benötigt wird – im Gegensatz zum ersten Verfahren. Die im k-ten Schritt gefundenen großen k-Sequenzen werden in Form der Paare ihrer Eltern gespeichert. Dadurch wird Speicherplatz gespart, da in der Praxis die großen k-Sequenzen am Ende der eigentlichen Arbeit des Verfahrens entstehen können, während des Durchgangs von der Menge L_1 zur Menge L_n und keinen überflüssigen Speicherplatz während der Arbeit des Hauptzyklus des Verfahrens einnehmen.

Schwierig kann sich der Prozeß des Füllens der benötigten Struktur gestalten, da während der Bearbeitung der Transaktion die Nummern der Kandidaten bekannt sind, nicht aber die der großen Sequenzen, welche erst nach der Bearbeitung aller Transaktionen bestimmt werden. Somit werden während der Bearbeitung in die erneuerte Struktur die Nummern der in ihnen enthaltenen Kandidaten eingetragen. Nach Bearbeitung aller Transaktionen der Eingangsmenge müssen die in der Struktur enthaltenen Nummern unter Berücksichtigung der Kandidaten, welche die Supportbedingung nicht erfüllten, korrigiert werden. Der Teil des Verfahrens, in welchem die Nummern korrigiert werden, ist in Abbildung 3 dargestellt.

```
m = 0; // Akkumulator der Korrektur
forall c ∈ C_k do begin          // für jeden Kandidaten
        if c.count>=minsup then c.correction = -m-1
        else m++;
end
```

Abbildung 3: Fragment des Verfahren zur Korrektur der Nummern.

Nach Ausführung dieses Fragments des Verfahrens ist für jeden Kandidaten der Wert der Korrektur bekannt, welcher von der Nummer des Kandidaten abgezogen werden muß, um die Nummer der großen Sequenz zu erhalten. In der Tat, um die Nummer der Sequenz aus der Nummer des Kandidaten zu erhalten, muß die Anzahl der den Test nicht bestandenen Kandidaten abgezogen werden, deren Nummern kleiner der Nummer des betrachteten Kandidaten sind. Die letzte Etappe der Erneuerung der Struktur der Transaktionen ist die Transformation der Nummern der Kandidaten in die Nummern der großen Sequenzen, gleichzeitig mit der Entfernung der Nummern der nicht-großen Kandidaten und der Sortierung.

Die Sortierung dient der Beschleunigung der Suche der paarweisen Sequenzen während der Bearbeitung der Transaktionsstrukturen, dass heißt, derjenigen Sequenzen, mit denen die gewählte Sequenz vereinigt werden kann. Ohne die Sortierung der Zeilen der Strukturen der Transaktion würde die Suche des zweiten Elternteils einen vollständigen Durchgang aller nachfolgenden Zeilen mit Prüfung der Bedingung der Vereinigung bedeuten. Mit Hilfe der Sortierung der Werte der Nummern der großen Sequenzen wird eine steigende Ordnung eingeführt und folglich ergibt sich die Möglichkeit, mittels Binärsuche in schnellster Weise einen Wert in der Zeile zu finden, mit welchem beginnend die Nummern der Eltern-Sequenzen gespeichert werden. Weiter wird ein geordneter Durchgang durch die Zeile ausgeführt, bis eine Nummer einer Sequenz größer der maximal möglichen Elternnummer für den gewählten ersten Elternteil gefunden wird. Experimentelle Daten haben gezeigt, dass die eingeführte Sortierung, welche nicht unbedingt für die Arbeit des Verfahrens notwendig ist, die Arbeit in den Fällen beschleunigt, in denen in der mit einer gewissen Transaktion verbundenen Struktur nacheinander einige sehr lange Zeilen enthalten sind. In der ersten Variante, ohne Sortierung, konnte ein deutlicher Abfall der Effektivität in einigen Teilen der Transaktionsmenge beobachtet werden, weswegen die Gesamtzeit der Arbeit des Verfahrens stark verlängert wird. Nach der Realisierung der Sortierung arbeitete das Verfahren auf solchen Daten um ein Mehrfaches schneller, abhängig von den konkreten Eingangstransaktionsmengen.

Während der Bearbeitung der Transaktionsstrukturen werden nacheinander die Nummern der großen Sequenzen ausgewählt, welche im vorhergehenden Schritt des Verfahrens gefunden worden sind und welche in der laufenden bearbeiteten Transaktion enthalten sind. Für jede

gewählte Nummer entsteht die Aufgabe der Bestimmung der Nummern des ersten und des letzten Elternteils, mit denen die Sequenz mit der gewählten Nummer vereinigt werden kann. Außerdem muß für jeden Kandidaten die Nummer bekannt sein, um den Zähler des Supports erhöhen zu können. Teilweise wird diese Aufgabe wie in der ersten Variante des Verfahrens gelöst. Jedoch wird in der Realisierung der zweiten Variante eine leicht abgewandelte Idee benutzt, unter Berücksichtigung der Notwendigkeit, die Nummer des Kandidaten zu kennen. In der ersten Variante ist die Nummer des Kandidaten bekannt, da im Hauptzyklus faktisch ein Durchlauf durch die Kandidatenmenge stattfindet und zur schnellen Ermittlung der ersten und letzten Nummer möglicher Eltern wird in jedem Schritt eine spezielle Struktur erneuert. In der zweiten Variante wird eine modifizierte Struktur angewandt, unter Berücksichtigung der Notwendigkeit der Speicherung der Nummern der entsprechenden Kandidaten. Außer den Nummern möglicher Eltern wird in jedem Eintrag dieser Struktur gleichfalls die Nummer des ersten Kandidaten, welcher erzeugt werden kann, gespeichert. Auf diese Weise können über die Nummer des zweiten Elternteils einer großen Sequenz gleich die Nummern der Eltern ermittelt werden, mit welchen die betrachtete Sequenz vereinigt werden kann, sowie die Nummern der entsprechenden Kandidaten.

Die zweite Variante des Verfahrens erfordert, wie auch die zweite Etappe des ersten Verfahrens, die Allokation von Speicher für die Werte der Zähler des Supports und das Array der Nummern der Transaktion, welche als letzte den Zähler inkrementierte.

(7) Beschreibung des Programminterfaces

Das Programminterface wird in Form eines Beispiels der Benutzung des Verfahrens in einem Java-Programm betrachtet.

Die Klasse *TransactionSetSeq* wird zur Speicherung des Satzes der sequentiellen Eingangstransformationen benutzt. Eine Instanz der Klasse wird mittels folgender Anweisung realisiert:

TransactionSetSeq tss = new TransactionSetSeq();

Nun fügen wir alle Transaktionen des Eingangssatzes hinzu:

for(int i=0;i< <Anzahl Transaktionen>;i++) {

Jede sequentielle Transaktion wird als eine Instanz der Klasse *ItemSetSeq* repräsentiert:

ItemSetSeq iss = new ItemSetSeq();

Die Elemente der Transaktionen sind ganze Zahlen – zum Hinzufügen eines neuen Elementes zur Transaktion dient folgende Methode der Klasse *ItemSetSeq*:

void addItem(int item);

Wir fügen alle Elemente der aktuellen Transaktion hinzu:

for(int j=0;j< <Anzahl der Elemente der aktuellen Transaktion>; j++)

iss.addItem(<Nummer des Elementes>);

Zum Hinzufügen einer Transaktion zum Transaktionssatz wird folgende Methode der Klasse *TransactionSetSeq* benutzt:

void addTransaction(ItemSetSeq iss);

tss.addTransaction(iss);

}

Zur Berechnung der Sequenzen muß eine Instanz der Klasse *Sequential* oder *Sequential2* erzeugt werden:

Sequential seq = new Sequential();

Sequential2 seq2 = new Sequential2();

Zum Starten des Verfahrens ist es notwendig, folgende in den Klassen *Sequential* und *Sequential2* enthaltene Methode aufzurufen:

SequentialResult SequentialAlg(TransactionSetSeq tss, double minsupp);

wobei

tss – Eingangsmenge der Transaktionen,

minsupp – minimaler Supportwert.

SequentialResult seqResult = seq.SequentialAlg(tss,minsupp);

oder

SequentialResult seqResult = seq2.SequentialAlg(tss,minsupp);

Das Ergebnis der Arbeit des Verfahrens wird in dem Objekt der Klasse *SequentialResult* gespeichert.

Zum Zugriff der gefundenen Sequenzen enthält die Klasse *SequentialResult* folgende Methode:

ItemSetSeqList getLargeSequences();

ItemSetSeqList issList = seqResult.getLargeSequences();

Die Klasse *ItemSetSeqList* enthält die Methoden:

int getSize(); – liefert die Anzahl der gefundenen Sequenzen,

ItemSetSeq getItemSetAt(int index); – liefert die Sequenz der Nummer *index*.

for(int i=0;i < issList.getSize(); i++) {

ItemSetSeq iss = issList.getItemSetAt(i);

Zum Zugriff der Sequenz enthält die Klasse *ItemSetSeq* folgende Methode:

int getItemAt(int index);

Folgende Methode liefert die Länge der Sequenz:

int getSize();

for(int j=0;j<iss.getSize();j++)
int item = iss.getItemAt(j);
}

Den Wert des Zählers für den Support liefert folgende Methode der Klasse *ItemSetSeq*:

int getSupportCount();

Der eigentliche Supportwert *support* für jede Sequenz ist der Quotient des Wertes des Support-Zählers und der Anzahl aller Transaktionen.

(8) Anwendung

Im weiteren soll die Anwendung des beschriebenen Verfahrens am Beispiel der Clickstreamanalyse von Webservern erläutert werden.

Webserver speichern die Informationen über abgerufen HTML-Seiten einer Website in Logfiles oder Datenbanken. Die Information über die abgerufenen HTML-Seiten wird hierbei in einer Tabelle gespeichert. Die Spalten entsprechen den Informationen über die abgerufene Seite wie beispielsweise Datum und Uhrzeit, URL der aufgerufenen HTML-Seite, Referer-URL, Typ des Web-Browsers. Die Zeilen entsprechen den abgerufenen HTML-Seiten selbst. Über verschiedene Mechanismen, die hier nicht weiter ausgeführt werden sollen, ist es möglich, *Websessions* zu identifizieren und jeder abgerufenen Seite zuzuordnen. Eine Websession ist eine Sequenz der von einem User der Website hintereinander abgerufenen HTML-Seiten.

Aufgabe der Clickstreamanalyse ist das Herausfinden typischer Pfade, entlang derer sich die User einer Website hauptsächlich bewegen. Beispielsweise kann es für einen E-Commerce Shop passieren, dass viele Nutzer über die Hauptseite den Shop betreten, in eine gewisse Produktkategorie gehen, ein spezielles Produkt in Detailansicht betrachten und danach - den Shop verlassen. Das lässt darauf schließen, dass das betrachtete Produkt die Erwartungen verfehlte und seine Präsentation dringend verbessert werden muß. Wenn Nutzer zyklische Pfade wählen, lässt sich oftmals daraus schließen, dass sie der Shop unübersichtlich aufgebaut ist, usw.

Das Herausfinden der typischen Pfade in Websites stellt eine Aufgabe der Sequenzanalyse dar. Hierbei ist $I = \{i_1, i_2, ..., i_n\}$ die Menge aller HTML-Seiten unserer Website. Jede Transaktion T entspricht einer Websession, deren Elemente eine Sequenz $<s_0, s_1, ..., s_k>$ von HTML-Seiten bilden. Die Gesamtzahl der Transaktionen D entspricht allen Websessions des

Webservers. Die Aufgabe der Clickstreamanalyse besteht nun darin, für einen vorgegebenen minimalen Supportparameter s alle großen Sequenzen von HTML-Seiten zu finden. Diese stellen die typischen Nutzerpfade der Website dar, deren Analyse - wie oben dargelegt - zu einer Verbesserung der Struktur der Website genutzt werden kann.

Zur Berechnung der großen Sequenzen ist das in der Erfindung beschriebene Verfahren sehr gut geeignet, insbesondere deswegen, weil es große Datenmengen analysieren kann. Gerade Logfile-Tabellen stark besuchter Websites weisen aber häufig Millionen abgerufener HTML-Seiten auf und stellen somit höchste Ansprüche an Sequenzanalyseverfahren. Die Auswertung dieser Informationen mittels Sequenzanalyse war daher zumeist außerordentlich rechenzeitintensiv und auf Sonderfälle (Sequenzen der Länge 2, nur wenige Websessions, usw.) beschränkt. Das beschriebene Verfahren überwindet diese Beschränkung für nahezu alle auf derzeitigen Webservern anfallenden Clickstream-Daten.

Die in der vorstehenden Beschreibung offenbarten Merkmale der Erfindung können sowohl einzeln als auch in beliebiger Kombination für die Verwirklichung der Erfindung in ihren verschiedenen Ausführungsformen von Bedeutung sein.